Ankica Barišić

Master of Science

Usability Evaluation of Domain-Specific Languages

Dissertação para obtenção do Grau de Doutora em

Informática

Orientador: Vasco Miguel Moreira Amaral, Assistant Professor,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Co-orientador: Miguel Carlos Pacheco Afonso Goulão, Assistant Professor,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Júri

Presidente:	Prof. Doutor Nuno Manuel Robalo Correia
Arguentes:	Prof. Doutor Antonio Vallecillo
	Prof. Doutor João Carlos Pascoal Faria
Vogais:	Prof. Doutora Ana Maria Diniz Moreira
	Prof. Doutor Marjan Mernik
	Prof. Doutor Vasco Miguel Moreira do Amaral

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

December, 2017

Usability Evaluation of Domain-Specific Languages

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I thank my parents, sister, brother and rest of my family, I am grateful for the love that they share and continuous support. Without them, nothing would be possible. I would like to show my gratitude to all of my friends which stand close to me during my doctoral program, and to my Capoeira family which helped me to become independent and to overcome the cultural difference.

My sincerest gratitude and appreciation goes to my professors, mentors, and advisers, Dr Vasco Amaral and Dr Miguel Goulão. I am truly indebted for Dr Amaral's and Dr Goulão involvement in my academic, professional, and social lives for last years. Not only they shepherded me in research, teaching, and outreach activities, but also gave me an outstanding example, how to become a computer science researcher while keeping a social consciousness.

I would like to show my gratitude to my NOVA-LINCS research colleagues, which supported me continuously during my research. Especially, to our research group leader Dr Ana Moreira, she was continuously supporting us to work together and share the experiences among colleagues.

To Dr Marjan Mernik, for his more involved role as a PhD committee member. I really appreciate his advice and direction on each of the key stages in my graduate study.

I would like to thank Prof. Ademar Aguiar for supporting me in a middle phase of my research by providing me with the opportunity to work and learn on Movercado project and helped me in creating new research ideas.

I am also indebted to the help and guidance I gained from the Dr Bruno Barroca, Dr Eugene Syriani, Dr Jeff Gray, Dr Hans Vangheluwe, Dr Dominique Blouin and Dr Jacome Cunha. I appreciate their guidance and assistance during my studies.

Finally, I would like to thank the financial support provided by the Department of Computer Science, of Universidade NOVA de Lisboa, to the department director Dr Luis Caires and my graduate program director Dr Nuno Correia. None of these can be accomplished without department support.

Abstract

The adoption of Domain-Specific Languages (DSLs) is regarded as an approach to reduce the accidental complexity of software systems development. The availability of sophisticated language workbenches facilitates the development of DSLs making them increasingly more popular. This comes at the risk that a badly designed DSL can bring more harm and decrease productivity, when compared to an existing alternative. In particular, a poorly designed DSL can be too hard to adopt by its domain users. As such, Usability is one of the key characteristics to mitigate this risk as it has an important impact on the achieved productivity of DSL users.

The current state of practice in Software Language Engineering (SLE) neglects the Usability of DSLs. A pertinent research question in SLE is how to systematically engineer Usability into DSLs. We argue that a timely systematic approach based on User Interface experimental evaluation techniques should be used to assess the impact of DSLs during their development process, while the cost of fixing the usability problems is relatively low when compared to fixing them at the end of the development process. For that purpose, the focus of this dissertation is to build a systematic approach that supports the iterative development process of DSLs concerning the issue of their Usability evaluation, and engages the DSL's end users in the process.

To be effective, the systematic approach should be grounded on the information produced along the engineering process. Model-Driven Development (MDD) enables us to explicitly capture the usability evaluation process by using models and establishing traceability links among them.

We propose the Usability Software Engineering Modelling Environment (USE-ME) as a conceptual framework for the usability evaluation of DSLs. We defined the evaluation process in a step by step manner. We demonstrated the feasibility of the conceptual framework building a USE-ME prototype to support it. USE-ME modelling instances provide decision support when determining the usability of the DSL and opportunities for its improvement. Finally, we conducted several case studies to illustrate the proposed conceptual framework.

Keywords: Domain-Specific Languages, Software Language Engineering, Experimental Software Engineering, Usability Engineering

Resumo

A adoção de linguagens específicas de domínio (DSLs) é considerada uma abordagem para reduzir a complexidade acidental do desenvolvimento de sistemas de software. A disponibilidade ferramentas recentes e sofisticadas de suporte ao desenvolvimento de linguagens ("modelling workbenches") tem tornado as DSLs populares. No entanto, esta popularidade tem colocado em evidência os riscos de uma DSL mal projetada. Uma DSL mal concebida pode causar mais danos e reduzir a produtividade, em comparação com alternativa existentes. De facto, uma DSL problematica pode ser muito difícil de adotar pelos seus utilizadores. Como tal, a preocupação com a Usabilidade é uma das principais características para mitigar esse risco, pois tem um impacto significativo na produtividade alcançada dos utilizadores da DSL.

A prática corrente da Engenharia de Linguages ("Software Language Engineering - SLE"), por se focar no desenho e implementação, negligencia a usabilidade de DSLs. Uma questão de investigação relevante em engenharia de Linguagens é de como integrar de forma sistemática no processo de engenharia da linguagem, a preocupação com a usabilidade. Nós argumentamos que, uma abordagem sistemática atempada baseada em técnicas de avaliação experimental da interface com o utilizador, deverá ser usada para avaliar o impacto das DSLs durante seu processo de desenvolvimento. Pretende-se que seja desenrolada enquanto o custo de corrigir os problemas de usabilidade é relativamente baixo em comparação com o que poderá acontecer em fases tardias do processo de desenvolvimento.

O foco desta dissertação é construir uma abordagem sistemática, que seja integrada com o processo comum de desenvolvimento de DSLs e que se foque na questão da avaliação de usabilidade envolvendo a todo o momento os utilizadores.

Para ser eficaz, a abordagem sistemática deve basear-se na informação produzida ao longo do processo de engenharia. O desenvolvimento orientado a modelos ("Model-Driven Development - MDD") permite-nos capturar explicitamente toda a informação relevante ao processo de avaliação usando modelos e estabelecendo links de rastreabilidade entre eles.

Nós propomos o Ambiente de Modelação de Engenharia de Software de Usabilidade (USE-ME) como uma estrutura conceptual para a avaliação de usabilidade de DSLs. Nesta

teses definimos, passo-a-passo, o processo de avaliação. Este trabalho demonstra a sua viabilidade tendo-se desenvolvido um protótipo do USE-ME. As instâncias de modelo em USE-ME, fornecem toda a informação necessária de suporte à decisão quanto à usabilidade da DSL, apontando para as oportunidades para a sua melhoria. Finalmente, iremos discutir detalhadamente vários casos de estudo realizados num ambiente empresarial e académico para ilustrar a metodologia proposta.

Contents

List of Figures

List of Tables

Acronyms

ACM	Association for Computing Machinery.
API	Application Programming Interfaces.
APN	Algebraic Petri-net.
COMLAN	Computer Languages Systems and Structures.
COST	European Corporation in Science and Technology.
DSE	Design Space Exploration.
DSL	Domain-Specific Language.
DSM	Domain-Specific Modeling.
DSME	Domain-Specific Modeling Environment.
DSM-TP	Domain Specific Modelling: Theory and Practice.
EMF	Eclipse Modeling Framework.
ESE	Experimental Software Engineering.
GME	Generic Modeling Environment.
GPL	General Purpose Language.
HCI	Human-Computer Interaction.
HEP	High Energy Physics.
IDE	Integrated Development Environment.
MDD	Model-Driven Development.
MPM	Multi-Paradigm Modeling.
MPM4CPS	Multi-Paradigm Modeling for Cyber-Physical Systems.
MVC	Movercado.

Pheasant	Physicist's Easy Analysis Tool.
PSI	Public Service International.
RPG	Role-Playing Game.
RSA	Rational Software Architect.
SAC	Symposium On Applied Computing.
SLE	Software Language Engineering.
SRC	Student Research Competition.
UCD	User-Centered Design.
UI	User Interface.
UML	Unified Modeling Language.
USE-ME	Usability Software Engineering Modelling Environment.

Chapter 1

Introduction

1.1 Context and motivation

The increasing pace at which the software is adopted in daily tasks, including those of users not necessarily proficient with computing, is pushing the need for rapid production of a growing number of complex software applications. The degree of specialisation in certain areas is pushing for the involvement of domain concepts in the software development process, as complex software configuration tasks. A **Domain-Specific Language (DSL)** is specialised in a particular application domain [148]. It can be defined as a user empowerment tool to increase productivity in software systems development [12, 113]. It offers the expressiveness required to specify the software applications at a higher level of abstraction, after which they can be automatically deployed or even simulated, with notations closer to the end user. DSLs are designed to bridge the gap between the problem domain (essential concepts, domain knowledge, techniques, and paradigms) and the solution domain (technical space, middleware, platforms and programming languages) [204]. Bridging this gap is expected to increase language users' productivity.

Practitioners often experience some practical difficulties when adopting DSLs [87]. During the language development, the importance of aligning the DSLs with the needs of their end users seems to be underestimated [112, 202]. The necessity for assessing the impact of introducing a DSL in a domain workflow has been discussed in the literature, often with a focus on the business value that DSL can bring [113]. This business value often translates into productivity gains resulting from the extent to which the domain users can use the DSL in practice [204].

Although building and adopting DSLs may seem intuitive, we need to have means to evaluate their impact. The measure of success has to be determined by assessing the impact of using DSL, in a realistic context of use, by its target domain users [17].

Investment into this assessment, commonly called usability evaluation, is justified by a reduction in development costs and increased revenues for other software products, brought by an improved effectiveness and efficiency by their end users [39, 139]. We expect a similar effect by introducing this practice into DSL development.

The software industry does not often report investment on the assessment of DSLs [76, 122]. Most of the reported DSL evaluations are performed only at final stages of a development cycle when changes in the DSL have a significant impact on the budget. The lack of systematic approaches, guidelines and comprehensive set of adequate tools may explain this shortcoming in the current state of practice. We argue that this situation is due to the perceived high costs of DSL evaluation, which lacks a consistent and computer-aided integration of two different and demanding complementary software processes: DSL development and usability engineering.

Software Language Engineering (SLE) is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages [118]. Although the phases of the DSL life cycle are systematically defined [148], we claim that this process lacks one crucial step, namely language evaluation [19]. Existing **Experimental Software Engineering (ESE)** techniques [32] combined with **Usability Engineering** techniques [155] can be adopted to support the DSLs' evaluations. Our goal is to *promote quality in use of DSLs by building up a conceptual framework that supports their development process by leveraging usability as a first-class concern*.

We can engineer DSLs to become more usable with a combination of both proactive and reactive approaches. Current proactive approaches that can be used to improve DSL usability, such as guidelines for developing visual notations [151], usability heuristics [154], cognitive dimensions of notations [41, 88], or even quality assessment framework for DSLs [110]. Reactive approaches are necessary as well [122]: DSLs should be tested experimentally with users using systematic techniques to confirm the impact of design decisions in a real context of use. In the early stages of this dissertation work, we highlighted the experimental approach [20], based on four DSL evaluation experiments that are examples of best practices ([18, 116, 125, 152]). Recently, we can find more examples of performing this kind of assessments in practice (e.g. [2, 68, 96, 106, 145]).

Usability concerns need to be addressed from the early stages of the DSL life cycle so that practitioners can perform timely evaluations [51]. Rather than designing the complete DSL before the implementation, abstractions should be evaluated iteratively and incrementally, in the context of a development cycle [17]. Building a systematic iterative usability evaluation approach is supposed to mitigate the risk of producing the inappropriate solutions that often cannot be reused. This work is expected to enhance the community's awareness to the relevance of DSLs' usability assessments to bridge the gap between the domain users and abstractions provided by language engineers.

1.2 Research problems

Our research work tackled the following problems:

- **Absence of a systematic approach for DSL usability evaluation.** The current state of the art does not report on existing systematic approaches for DSL evaluation. As will be detailed in Chapter 3, this absence leads to several problems: lack of integration of DSLs with other software engineering processes, absence of the effectiveness measures of DSL approaches, no proofs for applicability of DSL in targeted domain, among others. Some researchers already highlighted this issue and are looking for alternative approaches.

- **Promoting usability concerns since an early stage of development of DSLs is perceived as expensive.** The lack of a systematic evaluation approach, the involvement of domain experts rather than domain users in the development process, and the diversity of domains, and therefore of domain users which the corresponding DSLs are meant to support, are some of the main reasons that make the usability assessment perceived as expensive and not reusable. For instance, due to this inability to express best practices, most of the performed evaluation studies are not mature enough (toy examples).

- **Lack of Integrated Development Environment (IDE) support for the development of DSLs with high Quality in Use.** In order to support language engineers with a systematic usability evaluation approach, it is necessary to provide an approach supported by adequate tools which can be integrated with existing IDE support for DSL development.

The results of the preliminary systematic literature review of Gabriel et al. [76] and more recently by the systematic mapping study of Kosar et al. [122], share a particular concern regarded to the clear lack of DSL evaluation efforts reported in the research. In particular, there is an urgent need for controlled experiments supporting these evaluations to become more common. There is also documented evidence that the problems tackled within this dissertation are real industry problems [111].

The impact of an evaluation process for DSLs is interesting from an industry point of view. With many organisations developing their languages, or hiring companies to develop such languages for them, our framework can aid them in conducting a usability evaluations and reaching more usable languages.

The main challenges while addressing the problems identified in this dissertation were:

1. **Defining an appropriate experimental model for DSL evaluation.** Most of the existing evaluations are performed ad-hoc, not reporting enough details of the experimental design or result analysis. This research work produced a general

experimental evaluation model, tailored for DSLs' experimental evaluation, and instantiates it in several DSL's evaluation examples [20] (Section 4.1).

2. **Integrating the usability evaluation process with the DSL development.** Both the usability evaluation and DSL development process are complex and evolving. Therefore, we discussed the quality criteria and proposed a development and evaluation process that can be used to achieve usable DSLs in a better way [17] (Section 4.2 and Chapter 7). By allowing significant changes to correct DSL deficiencies along the development process instead of just evaluating at the end of it (when it might be too late), the UCD was introduced, as it is claimed to reduce development and support costs, and reduce staff cost for employers [14].

3. **Applying the usability assessments to DSL development in industrial context and for different domains.** The DSLs are developed for different domains, each of them having the users with a different background knowledge and necessity to understand specific concepts. It was challenging to apply our approach in the development of DSLs for different domains and industrial contexts. However, we succeeded to apply our approach in the context of the following domains: High Energy Physics (HEP) [18] (Section 9.1), model merge approach [15] (Section 9.5), humanitarian campaign management [23] (Section 9.3) and low-cost robotics for children (Section 9.4). In the case of the last two studies, we followed several iterative cycles of development to observe the impact of our approach during a long-term process.

4. **Developing the conceptual framework and tool support.** It was challenging to capture the complexity of information and process and leverage it in a systematic fashion, as presented in Chapter 5. We developed a tool support [27] (Chapter 6) which helps to use discovered knowledge, validate assumptions and provides a possibility to automate. This knowledge is presented in a formal model which captures only the meaningful information, helps with traceability and development decisions.

1.3 Research questions

The objective of the research is *to promote quality in use of DSLs by building up a conceptual framework that supports their development process, leveraging usability as a first-class concern*. This will involve the integration and adaptation of the current evaluation methodologies, their concepts, methods, tools, processes, and metrics.

As briefly presented in Section 1.2, the aim of our research consists in providing contributions for the following major problems faced in the realm of DSL usability evaluation:

RP1: Absence of a systematic approach for the DSL usability evaluation.

RP2: How to promote usability concerns since an early stage of development of DSLs.

RP3: Integration of systematic approach with existing IDE support for development of DSLs built with high Quality in Use.

In particular, we address the following research questions:

RQ1: How to model the DSL usability evaluation?

RQ2: How to promote usability concerns from an early stage of development of the DSL?

RQ3: How to integrate the proposed systematic approach to build usability evaluation in the development process of the DSL?

The RQ1 is related with RP1, RQ2 with RP2 and RQ3 is related with RP3. Each of the above research questions is also related to the research hypotheses in the following section.

1.4 Research approach

The motivation of this work is to **provide a systematic methodological approach to evaluate the usability of domain-specific languages during its development for application domains targeting large end-user groups**. The design science methodology [182, 206, 210] fosters the creation of artefacts that are driven to problem-solving projects. Wieringa [209] regards design science projects as a set of nested regulative cycles that solve practical (i.e. engineering) and knowledge (i.e. research) problems that are decomposed in subproblems. An engineering problem is defined as a *"difference between the way the world is experienced by the stakeholders and the way they would like it to be"* and a research problem is the *"difference between the current knowledge of stakeholders about the world and what they would like to know"*.

1.4.1 Addressing the engineering problem

We argue that, in general, one of the main goals of any DSL that is meant to be used by humans is to improve the productivity of its user; Thus, we provide the Usability Software Engineering Modelling Environment (USE-ME) framework [26] (Chapters 5 and 6) as a solution to an engineering (i.e. practical) problem. The regulative cycle follows the five tasks: problem investigation, solution design, design validation, solution implementation and implementation evaluation (see Figure 1.1).

1.4.1.1 Problem investigation

The first step was to analyse the problem in detail during the 'Problem investigation'. We started by focusing our attention onto stakeholders who have the need for a DSL, i.e. Domain Users, and specify their goals regarding a development of the DSL (Chapter 2).

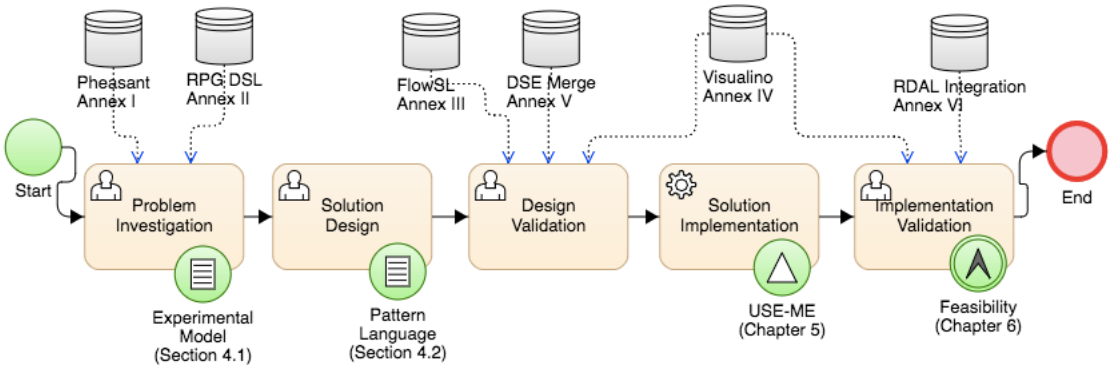


Figure 1.1: Research Process Overview

To understand better the problem they are facing and their causes, we have analysed the existing DSL development techniques and artefacts produced during a regulative development cycle. The development of different DSLs followed in the context of the FCT/UNL MSc courses on DSLs and the Domain Specific Modelling: Theory and Practice (DSM-TP) summer school series [1]. These activities gave us a practical experience in using different modelling workbenches (EMF, TextEdit, Eugenia, MetaEdit, Kaos, ATL). We reported our experience during the development of a Role-Playing Game (RPG) DSL following the regulative development cycle [141] (Section 9.2, Annex II). Besides that, we have performed an initial evaluation on the DSL named Physicist's Easy Analysis Tool (Pheasant) for the HEP domain [18] (Section 9.1, Annex I). This study helped us understand the impact of the problem in the context where the DSL is meant to be used by non-programmers in a sensitive context. Finally, we analysed existing evaluation examples of DSLs and specified a generic experimental model [20] (Section 4.1). The experimental model helped us understanding the criteria for stakeholders to consider the problem as solved.

1.4.1.2 Solution design

The following step was to analyse available solutions and design new ones during 'Solution design'. It was necessary to research the domain to justify that none of the existing solutions solves the problem [19]. As no satisfactory solution has been found, there was room to propose a new one [17, 21] (Section 4.2, Appendix B). The usability of a language needs to be evaluated through controlled experiments involving the language's end users. To be able to identify potential quality problems that will lead to user interaction and experience problems, a suitable approach is to apply UCD practices during design and development of the language. However, this practices makes it hard to control budget and plan time and responsibilities accordingly. Therefore incremental, iterative process should be applied, which enables tracking of design changes and validation of usability metrics.

[1] dsm-tp.org (accessed September 19, 2017)

1.4.1.3 Design validation

Once this design is completed, it is necessary to validate the solution during 'Design Validation' before its realisation. For that matter, the solution properties are assessed according to the criteria defined in the problem investigation, characterising the context of the target application and the coverage of the solution. If the solution has the desired effect for stakeholders in their context, the solution can be finally implemented. For that purpose, we have applied the approach in the two industrial cases of DSL development, namely FlowSL [23] (Section 9.3, Annex III) and Visualino (Section 9.4, Annex IV). FlowSL served an instantiation of usability evaluation into a DSL development guided by agile management during three iteration cycles [13]. The second one, Visualino, was followed along three development iterations and evaluated after each cycle. This case study is a good representation of the DSL where the usability evaluation is mandatory as the users are children, and the programmed behaviour is expected to run on a physical system i.e. an Arduino robot. Finally, we applied our approach in the case of DSE Merge language [15] (Section 9.5, Annex V), which is to be used by the programmers. This experience showed how the end users who are programmers could also benefit from the usability approach. Also, in this familiar environment it was easier to set up a more complex experiment and explore several possibilities for reusing it a virtual evaluation environment.

1.4.1.4 Solution implementation

The next step was to perform the 'Solution implementation', which defines the concepts and activities which are mandatory for the application of the approach. The main concepts supporting the modelling approach for usability evaluations are formally specified as UML class diagrams. The flow of activities is described by UML activity diagrams. The supporting conceptual framework described in Chapter 5 presents a generic systematic approach specified in a formal model as a solution for the engineering problem addressed in the scope of this thesis.

1.4.1.5 Implementation evaluation

Finally, the 'Implementation Evaluation' was performed as follows:

- The specification of the USE-ME conceptual framework, using UML diagrams which define the abstract syntax, was reviewed by a research group of NOVA-LINCS during a presentation session followed by individual questionnaires.

- The feasibility of experimental models implementation was validated through the implementation of the prototype tool and an instantiation of the evaluation models for the Visualino case study (Section 9.4, Annex IV) in Chapter 6. This showed that it was possible to model a performed usability assessment following the predefined

steps. Afterwards, we performed a pilot empirical evaluation of the implemented tool on four DSL development projects [28]. We have shown that it is feasible to capture the relevant information for the evaluation and its use in the result analysis.

- The feasibility to integrate our approach with another language for supporting DSL development, was validated on the case of USE-ME integration with the RDAL requirements approach [25] (Section 9.6, Annex VI). This showed that it is feasible to integrate USE-ME approach with existing approaches.

- Finally, we systematically obtain a community feedback about the feasibility and usefulness of the conceptual framework proposed in the context of this thesis and document its model using USE-ME approach (Chapter 8). A detailed interview was run with people which used our approach both in early, or later phases of the DSL development process. An evaluation survey was also conducted within the DSL community.

1.4.2 Addressing the research problem

To address the knowledge (i.e. research) problem, the work has regularly been submitted to international conferences for peer-reviewing after each milestone of results gathering and discussion. During this research work, the obtained results were materialised into the publications presented in Table 1.1. They are grouped in three respective areas of interest: Systematic approach for the DSL usability evaluation, Thesis Proposal and Case Studies. Publications reflect the obtained results, even if preliminary, to obtain the recognition of the community about the relevance of the problem.

The research work on this topic led to peer-reviewed publications of approach in the Computer Languages Systems and Structures (COMLAN) Journal [26], a book chapter [20], at the conference on Pattern Languages and Programs (PLOP) 2012 [21], Multi-Paradigm Modeling (MPM) at IEEE/Association for Computing Machinery (ACM) MODELS 2011 [17] and Portuguese National Symposium on Informatics (INFORUM) 2011 [19]. Performed case studies were published in the ACM Symposium On Applied Computing (SAC) conference 2018 SLE conference 2017 [25] and workshop events, namely, Model-Driven Development Processes and Practices (MD2P2) at IEEE/ACM MODELS 2014 [23], Domain-Specific Modeling (DSM) at SPLASH 2012 [141], 2012), and Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2011 [18]. Finally, the research proposal was accepted at doctoral symposiums of two relevant conferences, namely, QUATIC 2012 [22] and MODELS 2013 [13], as well as at the ACM Student Research Competition (SRC) at ACM SPLASH 2017 [16] and ACM/IEEE MODELS 2013 [14]. These publications have obtained over 100 citations, indicating a relevance of performed work and acceptance of the approach in the research community.

Table 1.1: Publications

	Title	Year	Published by	C[2]
Systematic approach for the DSL usability evaluation				**57**
[26]	Usability Driven DSL development with USE-ME	2017	COMLAN Journal	1
[28]	USE-ME Empirical evaluation pilot study	2017	[Data set]. Zenodo.	
[27][3]	USE-ME 1.1	2017	[Data set]. Zenodo.	
[24]	Domain-Specific Language domain analysis and evaluation: a systematic literature review	2015	UNL, FCT, [Report]. Zenodo.	1
[20]	Evaluating the Usability of DSLs	2012	IGI Global	16
[21]	Patterns for Evaluating Usability of DSLs	2012	PLoP@SPLASH	11
[17]	How to reach a usable DSL? Moving toward a systematic evaluation	2011	MPM @MODELS	19
[19]	Quality in Use of DSLs: Current Evaluation Methods	2011	INForum	9
Case Studies				**47**
[29] (IV)[4]	Leveraging teens feedback in the development of a DSL: the case of programming low-cost robots	2018	SAC	
[25] (VI)	A Requirements Engineering Approach for Usability-Driven DSL Development	2017	SLE	
[15] (V)	STSM Report: Evaluating the efficiency in use of search-based automated model merge technique	2016	MPM4CPS @COST, [Report] Zenodo	1
[23] (III)	Introducing Usability Concerns Early in the DSL Development Cycle : FlowSL Experience Report	2014	MD2P2@MoDELS	5
[141] (II)	The RPG DSL: a case study of language engineering using MDD for Generating RPG Games for Mobile Phones	2012	DSM @SPLASH	11
[18] (I)	Quality in use of DSLs: a case study	2011	PLATEAU @SPLASH	30
Thesis proposal				**23**
[16]	Framework support for Usability evaluation of DSLs	2017	ACM SRC	
[14]	Iterative Evaluation of DSLs	2013	ACM SRC	
[13]	Evaluating the Quality in Use of DSLs in an Agile Way	2013	Doctoral Symposium @MoDELS	2
[22]	Usability evaluation of DSLs	2012	SEDES @QUATIC	21

Also, apart from productivity indicators regarding publications, the thesis candidate was invited to give seminars about the research, both abroad (University of Alabama, USA, 2013; University of Maribor, Slovenia, 2014; University of Malaga, Spain, 2016) and in Portugal (Agile & Scrum Portugal 2013). The candidate also contributed to the community in several roles, namely as a member of the teams organising the Domain Specific Languages: Theory and Practice (DSM-TP) summer school series from 2012, a DSL Summer Courses on Domain-Specific Languages in the University of Belgrade, Serbia, 2013-2014, as well as HuFaMo workshop @MODELS 2015. She was invited to be PC

[2]Citations obtained from https://scholar.google.pt/ on October 20 2017

[3]USE-ME tool - https://github.com/akki55/useme (accessed September 19, 2017)

[4]Visualino companion site - https://sites.google.com/view/vl-empiricalstudy/home (accessed September 19, 2017)

member of DSM at SPLASH since 2013, WAPL workshop at FedCSIS conference since 2015, Doctoral Symposium at IEEE/ACM MODELS 2014, and as a reviewer of the Journals ComSIS and ASE.J by Elsevier and SQJ by Springer. The candidate was awarded the following grants: Short Term Scientific Mission (STSM) for Multi-Paradigm Modeling for Cyber-Physical Systems (MPM4CPS) by European Corporation in Science and Technology (COST) IC1404 2016, SIGSOFT CAPS Award by ACM CAPS 2014, ACM-W Professional Activities Prize 2012 and ACM SIGPLAN Professional Activities Committee Prize 2011. Finally, the candidate is actively participating in EU ITC COST Action IC1404 MPM4CPS and DSML4MA TUBITAK/0008/2014 project for Developing a Framework for Evaluating DSM Languages for Multi-agent Systems.

1.5 Thesis outline

This thesis is divided into following major parts:

- context and related work
  - Chapter 2 deals with the problem definition. Here, we introduce the reader to the context of DSLs and their life cycle. It is followed by a description of the usability evaluation approaches and specification, and the motivation for our work.
  - Chapter 3 details the related approaches, and highlights the scope of related work, its benefits and shortcomings for solving the problem addressed in this thesis.
- proposal of systematic approach
  - Chapter 4 introduces concepts that are crucial for the argumentation of our proposed solution in the next part. The experimental model for DSL and patterns for evaluating usability of DSLs are described.
  - Chapter 5 presents a USE-ME conceptual framework as a proposed solution of the given problem.
- feasibility and applicability of proposed approach
  - Chapter 6 shows a feasibility of USE-ME approach and introduces a prototype tool.
  - Chapter 7 discusses applicability of USE-ME approach.
- evaluation of research questions and case studies
  - Chapter 8 presents a evaluation model for USE-ME.
  - Chapter 9 details the performed case studies.

Finally, Chapter 10 discusses future work and concludes the thesis.

Chapter 2

Research context

The immersion of computer technology in a wide range of domains leads to a situation where the users' needs become increasingly demanding and complex. Consequently, software engineers need to cope with the growth of both essential and accidental complexity [46]. They have to provide solutions that solve a class of crucial problems in a given domain, which is sometimes very difficult to learn, such as the rules and technical jargon found in knowledge areas such as Physics, Finance, Medicine, etc. Also, it is necessary to deal with the accidental complexity of the used technology, e.g., the use of low-level abstraction programming languages while integrating a wide plethora of different tools and libraries.The adoption of DSLs is regarded as an approach to reduce the accidental complexity of software systems development [61, 86, 205].

2.1 Domain-Specific Languages (DSLs)

A **DSL** is a language that supports solutions to essential problems from a given domain (e.g. Physics Computing, Financial Domain, Health-care, Control Systems, and Gaming). They are intended to raise the level of abstraction closer to users' domain understanding. Opposing to a General Purpose Language (GPL), such as Java or C++, that is meant to be applicable across domains, a DSL offers the end user the possibility to express his needs in terms of the domain of the problem instead of in terms of the computational solution [178]. DSLs provide a notation tailored towards an application domain as they are based on models of relevant concepts and features of the domain [148]. As DSLs are used to describe and generate members of a family of systems or products in the application domain, they give the expressive power to model the required family members more easily. DSLs are claimed to match users' mental model of the problem domain by constraining the user to the given problem [98]. Finally, DSLs simplify the development of applications

in specialised domains at the cost of their generality.

A DSL can offer several important advantages over a General Purpose Language (GPL) [63]:

- *Domain-specific abstractions:* a DSL provides pre-defined abstractions to represent directly concepts from the application domain;
- *Domain-specific concrete syntax:* a DSL offers a natural notation for a given domain and avoids the syntactic clutter that often results when using a GPL;
- *Domain-specific error checking:* a DSL enables building static analysers that can find more errors than similar analysers for a GPL and that can report the errors in a language familiar to the domain expert;
- *Domain-specific optimizations:* a DSL creates opportunities for generating optimized code based on domain-specific knowledge, which is usually not available to a compiler for a GPL;
- *Domain-specific tool support:* a DSL creates opportunities to improve any tooling aspect of a development environment, including, editors, debuggers, version control, etc.; the domain-specific knowledge that is explicitly captured by a DSL can be used to provide more intelligent tool support for developers.

The idea of DSLs is as old as the notion of programming languages [148]. Widely used DSLs are: Excel macro (spreadsheets), SQL (database queries), LaTeX (typesetting), HTML (hypertext web pages), VHDL (hardware design), PostScript, LabVIEW, Simulink, and Lego Mindstorms. They come in a wide variety of forms, e.g., textual, diagrammatic, graph-based, form-based, grid-based, etc. DSLs are also called: application-oriented, special-purpose, 4GL (4th generation), task-specific, problem-oriented, end-user or little languages.

2.1.1 Growing adoption of DSLs supported by MDD

The use of the **MDD** techniques and tools is seen as a viable approach for dealing with accidental complexity [205]. MDD is grounded on the notion of providing explicit Models, seen as 'first class artefacts' in the process, that are further transformed into other lower level, more detailed, models. These transformations are also considered as development artefacts and can be explicitly modelled by using transformation models. This approach has the special impact of dealing with the complexity of large-scale problems while enabling rapid prototyping, simulation, validation and verification techniques [113, 208].

In general, *'A model is a representation of something, constructed and used for a particular purpose'* [36]. Kühne defines this concept more appropriate to our context as *'A model is an abstraction of a (real or language-based) system allowing predictions or conferences to be*

made' [126]. Modellers build models to represent something. But a model has no meaning by itself. The information in the model can be understood if the model is combined with an interpretation. Extracting the correct meaning from the model can only be achieved if a common understanding of the concepts between modeller and the interpreter (who will typically use the model) is established. This common understanding leads us to a language. In practice, DSLs can be used to represent domain-specific models in MDD approaches. Each different model adopted by an MDD approach can be seen as a DSL that addresses the modelling of abstractions at a specific stage of software development. We have several guides that discuss how to implement a DSL based on MDD approach [11, 178, 187].

2.1.2 DSL implementation classification

DSLs can be implemented in different languages, such as textual or graphical languages, interactive GUIs, or embedded in other programming languages [89].

2.1.2.1 Internal vs. External DSLs

Fowler [74] and Gray et al. [86] handle DSLs in two different styles that can be distinguished with regard to the implementation approach of the DSL.

An *internal DSL*, also called as embedded DSL, is defined as an extension to an existing GPL and uses it as a base host language (e.g. Ruby DSLs [75] such as of first iteration version of FlowSL [23], or Hudak's embedded DSL [99]). It is not necessary to build a new generator for an internal DSL, as it can use the compiler or interpreter of the host language. DSLs can be layered on top of a host language using subtyping (i.e. programming libraries that define new classes with behaviours that reflect domain concepts). This layered style of DSL design is very unrestrictive because it does not preclude the use of non-DSL expressions. Implementation of a DSL via a definition of a new language from scratch is possible but is very restrictive because the language is self-contained.

An *external DSL* is defined in a different format than the host language of the application and transformed into it using some form of a compiler (e.g. Microsofts' OSLO framework or graphical DSLs, such as Pheasant [7], Lego [132], Visualino [135], etc.). An external DSL can use all kinds of language constructs as it enables designers to define any possible syntax independent from the syntactical particularities of a given host language.

2.1.2.2 Development without vs. with use of language workbench

The implementation of a DSL without a language workbench, e.g. coupled with its own development environment, is possible through rigorous planning and software engineering. In this case, an application with an interface for assessing the concrete syntax items of the language is the programming environment. This IDE of DSL design is also very restrictive, though it is important to note that the language definition is often obscured in the environment design, rather than decoupled from it.

A development with use of language workbench, i.e. Domain-Specific Modeling Environment (DSME), provides interfaces for activities such as expression building, model execution, and well-formedness checking (among others). This way to define a DSL involves the co-creation and synthesis of the structural portion (i.e., C, A, and Mc) of the DSL through the use of a meta-modelling environment. This style of DSL modelling design facilitates rapid development but is at the same time somewhat restrictive. It produces similar results to the IDE style of design, though it is significantly more sophisticated since the definition of the language is used to define the DSME, rather than a design-time result of the development of the DSME.

Different tools and platforms are now being defined to support DSL implementation and processing, such as, Microsoft DSL Tools [60], OpenArchitectureWare [93], Generic Modeling Environment (GME) [130], Rational Software Architect (RSA) [136] and Epsilon [120] based on GMF/EMF [90, 186], MetaEdit+ [193], AtomPM [189] and MPS [48].

2.1.2.3 Horizontal vs. Vertical DSLs

If we perform an analysis of the names of reusable components (in reusable infrastructures), and the reusable data structures and methods from existing Application Programming Interfaces (API), and determine the possibilities of how they can be composed in a meaningful way then we can infer a **bottom-up** DSL from that reusable infrastructure. This bottom-up method of building languages by reusing existing reusable infrastructures may, however, generate languages that lack generality in the capability of solving any class of problems of a given domain, or if the domain of the problem is not yet fully bounded (categorized), there may be irregular composition patterns that can be nonsense regarding the problem.

A **top-down** method would be to complete the domain analysis phase that is behind the existing reusable infrastructure, by discarding any existing implementation and focusing only on the complete description and categorization of the class of problems from which its users will use our new DSL to describe their solutions while using the identified problem concepts when regarded to its context of use. If we find a mapping between all the possible expressible solutions (which might be very difficult in some cases) in our new DSL and the existing concepts of a reusable infrastructure, then we have assembled a top-down DSL.

DSLs that are built in a top-down fashion are mostly called *horizontal DSLs* while DSLs that are built in bottom-up fashion are called *vertical DSLs* [118]. In practice, it is more common for a DSL design for human-computer communication to be built using a combination of bottom-up and top-down approaches.

2.1.3 DSL stakeholders

A **Language Engineer** is a professional who is skilled in the application of the engineering discipline to the creation of software languages. This professional manages the implementation priorities, designs the software language and is responsible for making the language functional at the system level. In general, Language Engineers are involved in the language specification, implementation, and evaluation, as well as in providing templates and scripts [118].

A **DSL User** or **Domain User** is any person who uses software languages to create applications (e.g. application developers) [118]. The possible user base of the models can easily be broader as domain-specific modelling allows application users to be better involved in the application development process. In that case, customers, other than typical application developers, can read, accept and, in some cases change application specifications, being directly involved in the application development process. A Domain User can work with models which apply concepts directly related to specific characteristics of the configuration, such as specifying deployment of software units to hardware or describing high-availability settings for uninterrupted services with redundancy for various fault recovery scenarios [113].

A **Domain Expert** is a person involved in the language development process, and is also sometimes known as a knowledge engineer. In the case of domain-specific modelling, they do not need to have the software development background, but they can specify the application for code generation. A Domain Expert specifies models for concept prototyping or concept demonstration, and Language Engineers can proceed from these models. They are responsible for managing system goals and iterations. In contrast with Domain Users, they should have domain knowledge that includes areas of all target model applications.

DSLs are usually built by Language Engineers in cooperation with Domain Experts [205]. In practice, Domain Users will use the DSL. These Domain Users are the real target audience for the DSL. The Domain Experts and Language Engineers can play the Domain User role, but they are, often, just a small subset of target end-users population. Although Domain Users are familiar with the domain, they are not necessarily as experienced as the Domain Experts. They may also lack the experience of Language Engineers in using languages. So, it may turn out that the language is valid by construction for Domain Experts and Language Engineers, but not necessarily to other Domain Users. Neglecting Domain Users in the development process may lead to a DSL they are not really able to work with.

2.1.4 DSL life-cycle

Several steps are detailed by Thibault [191], Völter [205], Mernik [148], Visser [201], Strembeck and Zdun [187], to develop a DSL, contributing to the formal definition of the DSL life-cycle. The following are listed as DSL development phases: decision, analysis,

design, implementation and deployment. Mernik [148] provided a set of patterns that describe common situations that potential developers may find themselves in, and were already tackled successfully by previous DSL development projects.

The first phase is the **Decision** that corresponds to the 'when' part of the DSL development, while the remaining phases correspond to the 'how' part. Its goal is to identify the need for a DSL to the domain and its validity, which includes justifying that the effort to invest in its creation can be compensated. To make the decision, the stakeholders (including Domain Experts and Language Engineers) need to discuss the requirements of the domain following DSL development patterns.

The following phase is the **Analysis**. Its goal is to define the domain model with support to the DSL. The analysis has to take into account the particularities of the domain that will be explored, such as the terms and expressions intrinsic to a problem. In this phase, Domain Experts helps Language Engineers to define the description of domain concepts, the feature models, the functional and technical requirements, and the goal model. The Analysis phase primarily produces a domain model, representing common and varying properties of the system within the domain [63]. The domain model will be used to assist with the creation of configurable architectures and components.

The next phase is **Design**, where Language Engineers formalise a language abstract syntax (i.e. meta-model) and define the representations for the model elements and production/composition rules. Additionally, the semantics of the language is defined. The design approaches of DSL design can be characterised along two orthogonal dimensions: the relationship between the DSL and existing languages, and the formal nature of the design description [148]. A DSL can be designed from scratch or based on an already existing language. Based on the reuse of existing languages there are three different design patterns: *piggyback* (is partially used), *specialization* (is restricted) and *extension* (is extended). The formal nature of DSL can range between:

- *informal* – a DSL is specified in natural language and/or with examples, and
- *formal* – a DSL is specified using one of the available semantic definition methods, e.g. regular expressions, grammars, etc.

The fourth phase is the actual **Implementation**, which includes the integration of DSL artefacts with the infrastructure, as well as the implementation of the necessary DSL to platform transformations. A DSL can be implemented by different approaches (e.g., interpreter, compiler, preprocessing, embedding, extensible compiler/interpreter, COTS, hybrid [148]), each having its own merits [123]. In this phase, the developers produce the model checkers and simulators, which will help the modeller to validate the specified models. Finally, the DSL is delivered with its documentation in the **Deployment** phase. While Domain Experts themselves can understand, validate and modify the software by adapting the models expressed in DSLs, more substantial changes may involve altering a

DSL implementation. Therefore, the DSL should have a migration strategy, as any other software product.

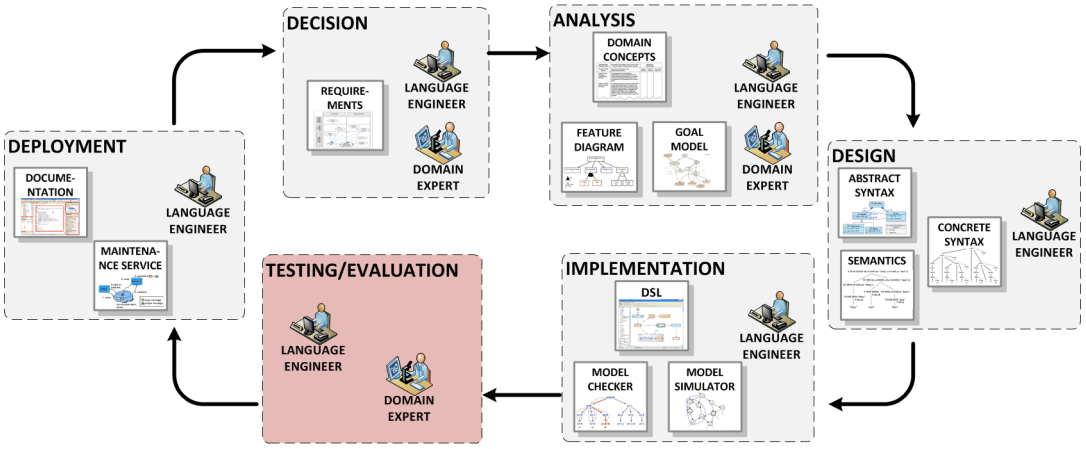


Figure 2.1: DSL Life-Cycle (taken from [26])

We argue that this process lacks an important step: **Evaluation** [17, 54], just before the deployment (see Figure 2.1), that should include the verification (testing if the right functionality is provided by the DSL) as well as its validation (testing if this DSL is right for its users). In the current state of practice, the focus of evaluation is only on the language engineering, and not on its usability, leading to a clear lack of validation involving the end-users [122]. Current verification is supported by model checkers and simulators. As the DSL promotes the modification of models which are claimed to be easier to produce and understand, the actual impact of the DSL in these tasks should be systematically evaluated involving its real users, performing real user actions. This phase is expected to help mitigate a pervasive problem of software engineering, i.e. software comprehension [166], that addresses the challenge for associating human oriented concepts with their counterpart solution domain concepts (e.g. computational terms).

Visser [201] recommends the inductive approach which, in opposition to designing the complete DSL before implementation, incrementally introduces abstractions that allow capturing a set of common programming patterns in software development for a particular domain. Visser also states that developing the DSL in iterations can mitigate the risk of failure. Instead of a big project that produces a functional DSL in the end, an iterative process produces a useful DSL early on. The availability of sophisticated language workbenches facilitates the development of DSLs making them increasingly more popular. This comes at the risk that a badly designed DSL can bring more harm than benefits and decrease productivity when compared to the existing baseline alternative (typically, a GPL, although a new DSL can also be developed to replace an existing DSL that is perceived not to be adequate enough for the goals of the organization using it). In particular, a poorly designed DSL can be too hard to adopt by its domain users. As such, to evaluate usability is one of the key strategies to mitigate this risk as usability has an important impact on the achieved productivity of DSL users.

2.2 Usability evaluation

DSLs are conceived as communication interfaces between human and computers. Therefore, if we take into account the main purpose of HCI, we can conclude that to evaluate DSLs has several similarities to assessing regular User Interface (UI)s [18]. We argue that any UI is a realisation of a language, where a language is considered as a theoretical object (i.e. model) that describes the allowed terms and how to compose them into the sentences involved in a particular HCI. On another hand, UI represents all points of human interaction (orchestrated inputs and outputs) with a DSL for solving a domain problem.

Usability Engineering is a field that is generally concerned with HCI and specifically with devising UIs that have high Usability. It provides structured methods for achieving efficiency and elegance in interface design [155]. Empirical (i.e. experimental) evaluation studies of UIs with real users is a crucial phase of the Usability engineering life-cycle [66]. A relevant set of quantitative and qualitative measurements must be inferred and combined together to lead to a useful assessment of the several dimensions that define software Quality in Use, often referred to as Usability [104].

2.2.1 Usability vs. Quality in Use

The notion of **Usability** is used in many different contexts and its definitions developed progressively. Usability is defined by Shackel and Richardson [180] as the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfil technically specified range of tasks, within the specified range of environmental scenarios. Shortly, Usability is *'the capability to be used by humans easily and effectively, where 'easily = to a specified level of subjective assessment'; 'effectively = to a specified level of (human) performance'*.

Usability is the quality characteristic that measures the ease of use of any software system that interacts directly with an end user. It is a subjective non-functional requirement that can only be measured indirectly by the extent to which it satisfies its corresponding needs based on the users' cognitive capacity. It focuses on features of the HCI. Usability is the result of the achieved level of quality in use of a software system i.e. a user's view of quality. It is dependent on achieving the necessary external and internal quality that is influenced by the achievement of different quality attributes dependent on a context of use. Tests of language Usability are based on measurements of the users' experiences with it.

International Organisation for Standardization (ISO) firstly defines Usability as 'the effectiveness, efficiency, and satisfaction with which specified users achieve specified goals in particular environments' (ISO 9241-11 [102]). Later on, the notion of Usability is integrated into the software quality framework under the term Quality in Use (ISO 9126 [103]). Finally, Usability is defined as *'degree to which a product or system can be*

used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use'. Usability can either be specified and measured as a product quality characteristic in terms of its sub-characteristics or specified and measured directly by measures that are a subset of quality in use (ISO/IEC 25010 [104]) (see Figure 2.2).

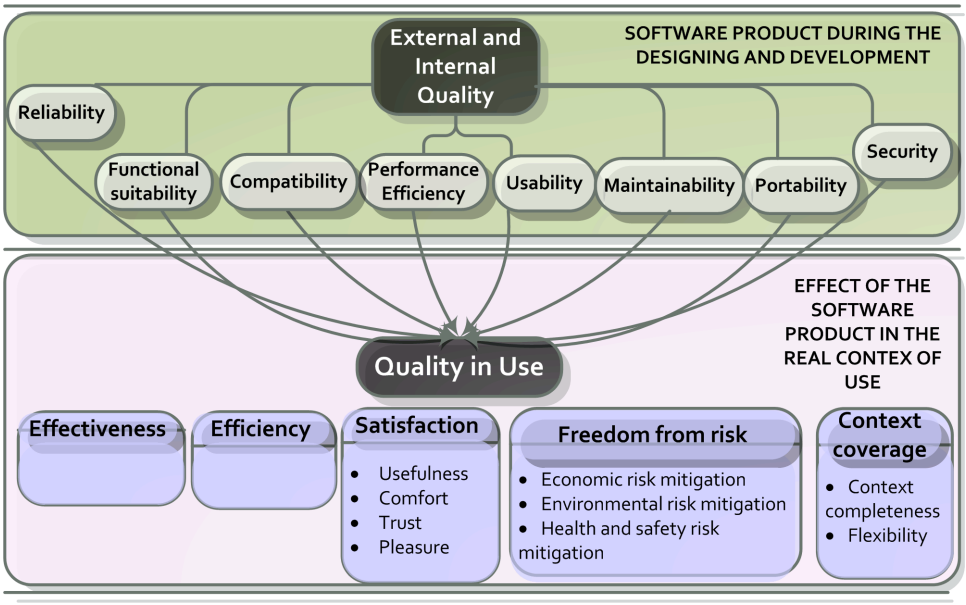


Figure 2.2: Quality in Use model by ISO/IEC 25010 [104]

Similarly to the other software qualities, Usability evaluation should not be simply added at the end of the development process. Instead, it has to be included in the development process from the beginning by taking into consideration internal and external quality attributes.

Internal quality is the *'totality of characteristics of the software product from an internal view'* that provides Usability metrics that are used for predicting the extent to which the software in question can be understood, learned, operated, attractive and compliant with Usability regulations and guidelines. Internal metrics can be applied to a non-executable software product during designing and coding. Internal metrics provide users, evaluators, testers, and developers with the benefit that they are able to evaluate software product quality and address quality issues early before the software product becomes executable [103].

External quality is the *'totality of characteristics of the software product from an external view'* that provide us with metrics that use measures of a software product derived from measures of the behaviour of the system of which it is a part, by testing, operating and observing the executable software or system. Before acquiring or using a software product it should be evaluated using metrics based on business objectives related to the use, exploitation and management of the product in a real Context of Use. External metrics provide users, evaluators, testers, and developers with the benefit that they are able to evaluate software product quality during testing or operation [103].

Evaluating **Quality in Use** validates software quality in specific user-task scenarios. *Quality in Use* is the user's view of the quality of a system containing software, and is measured in terms of the result of using the software, rather than properties of the software itself. Achieving *Quality in Use* is dependent on achieving the necessary *External quality*, which in turn is dependent on achieving the necessary *Internal quality*. Measures are normally required at all three levels, as meeting criteria for internal measures is not usually sufficient to ensure achievement of criteria for external measures, and meeting criteria for external measures is not usually sufficient to ensure achieving criteria for *Quality in Use*.

Achieving *Quality in Use* for different users means achieving different goals: for *the end user*, quality in use is mainly a result of Functionality, Reliability, Operability and Efficiency; for *the person maintaining* the software, quality in use is seen as result of Maintainability; for *the person porting* the software, as result of Portability. The new model for achieving *Quality in Use* provides a framework for a more comprehensive approach to specifying Usability requirements and measuring Usability taking into account the stakeholder's perspective.

To evaluate the achieved Quality in Use of DSLs we find it necessary to assess the following attributes, on which we concentrate in the scope of this thesis:

- *Effectiveness* should determine the accuracy and completion of the implementation of the sentences;
- *Efficiency* tells us what level of effectiveness is achieved at the expense of various resources, such as mental and physical effort, time or financial cost, and is commonly measured in the sense of time spent to complete a sentence;
- *Satisfaction* captures freedom from inconveniences and positive attitude towards the use of the language; and
- *Context coverage* with an emphases on learnability and memorability of the language terms.

There is an increasing awareness of the quality in use of languages, fostered by the competition of language providers. Better Usability is a competitive advantage, although evaluating it remains challenging. While evaluating competing languages, it is hard to:

- interpret the existing metrics in a fair, unbiased way;
- provide relevant design changes; and
- assure that the scope of their evaluation is preserved to target user groups.

Software **Expert Evaluators** typically implement complex experimental evaluation studies. Their expertise is essential to design the evaluation sessions properly and to gather, interpret, and synthesise significant results. Although it is desirable to have an

Expert Evaluator within the teams, it is not always possible. This calls for the need of automated tools that support these experts, as well as other DSL stakeholders.

2.2.2 Usability evaluation methods

The importance of Usability as a quality attribute has led to the development of several Usability evaluation methods, whose purpose is to determine systematically the degree to which a software product is easy to use. According to Fernandez et. al., [70], the Usability evaluation methods can be defined as *'procedures composed by a series of well-defined activities to collect data related to the interaction between the end user and a software product, in order to determine how the specific properties of a particular software contribute to achieving specific goals'*. Usually, these methods are employed during all phases of the software development process to ensure the design of a usable product that can meet high-quality standards.

Nielsen and Molich proposed evaluating Usability in four ways [154];

Formally by some analysis techniques. *Evaluations using models and simulations* can predict measures such as time to complete a task or the difficulty of learning to use a product. Some models have the potential advantage that they can be used without the need for any prototype to be developed.

Automatically by a computerized procedure. This can be done by *Automated checking of conformance to guidelines and standards* or by *Evaluation of data collected during system usage*. This kind of evaluation is possible when initial prototypes or initial versions of the full implementation are available.

Empirically by experiments with test users. *Evaluation with users* is recommended at all stages of development if possible, or at least in the final stage of development. Nielsen suggests at least eight participants per group in empirical evaluation for obtaining a significant number of participants [154]. Although this is only a rule of thumb, a pragmatic sample size that Nielsen has found convenient to provide 'good enough' results, rather than a sound sample size determination for the adequate number of participants in an evaluation. We can use:

- *Formative* methods that focus on understanding the user's behaviour, intentions and expectations in order to understand any problems encountered, and typically employ a 'think-aloud' protocol or

- *Summative* methods that measure the product Usability, and can be used to establish and test user requirements. Testing may be based on the principles of standards and measure a range of Usability components. Each type of measure is usually regarded as a separate factor with a relative importance that depends on the Context of Use. Iterative testing with small numbers of participants is preferable, starting early in the design and development processes.

Heuristically by simply looking at the product and passing judgment according to an own opinion. It is usually considered as *Evaluation conducted by expert* and it can be

Table 2.1: Frequency of Use of Each Usability Evaluation Method according to [162]

Usability Evaluation Method	**Percentage**
Survey / Questionnaire	26.26
User Testing	14.14
Heuristic Evaluation	12.63
Interview	10.35
User Testing – Thinking Aloud / Thinking Out Loud	9.6
Software Metrics / Usability Metrics	4.8
Automated Evaluation via Software Tool	4.04
Cognitive Walkthrough	2.78
Prototype Evaluation	2.78
Focus Group	1.52
Checklist Verification	1.26
Pencil & Paper	1.26
Perspective Based Usability Inspection	1
Field Observation / Field Study	<1
Eye Tracking	<1
Click Map / Scroll Map / Heat Map	<1
Opinion Mining	<1
Web Usability Evaluation Process	<1
Retrospective Thinking Aloud	<1
Cognitive Task Analysis	<1
Usability Guidelines	<1
Card Sorting	<1
Canvas Card Sorting	<1
Retrospective Sense Making	<1
Personas	<1
User Workflow	<1
Cognitive Jogthrough	<1
Domain Specific Inspection	<1
Participatory Heuristic Evaluation	<1
Semiotic Inspection Method	<1
Usability & Communicability Evaluation Method	<1
Simplified Pluralistic Walkthrough	<1
Simplified Streamlined Cognitive Walkthrough	<1
Music Performance Measurement Method	<1

used when initial prototypes are available. Expert methods that do not use task scenarios are referred to as reviews or inspections, while task-based evaluations are referred to as walkthroughs. Conducting expert evaluation is recommended to identify as many Usability issues as possible in order to eliminate them before conducting user-based evaluations.

In June 2015, Paz and Pow-Sang [162] performed a systematic mapping review of Usability evaluation methods on publications since 2012. They identify what are the

most widely used techniques to evaluate the Usability of software products in the context of a development process (See Table 2.1). Additionally, the review authors identify which are methods commonly used for each category of software application involved in a development process. The definitions for different evaluation methods according to Nielsen [155] and Paz et. al., [161] are provided in Appendix A. However, because of the broad range of these techniques, the choice of the most suitable method for a particular scenario has become a difficult decision. There is no agreement on what the best method is.

2.2.3 Usability design - how and when?

Usability has two complementary roles in design: as an attribute that must be designed into the product, and as the highest-level quality objective, which should be the overall purpose of design [163]. Two important issues are how and when to assess DSL Usability.

Concerning the **how**, we can think of DSLs as communication interfaces between their users and a computing platform, making DSL Usability evaluation a particular case of evaluating UIs [19]. This implies identifying the key quality criteria from the perspective of the most relevant stakeholders, to instantiate an evaluation model for that particular DSL [21, 110]. These criteria are the evaluation goals, for which a set of relevant quantitative and qualitative measurements must be identified and collected. We borrow from UI evaluation several practices, including obtaining these measurements by observing or interviewing, users [172]. In general, it is crucial that the evaluation of HCIs includes real users [66], for the sake of its validity. In the context of DSLs, the 'real users' are the Domain Users (see Section 2.1.3).

Interactive Usability investigation methods apply contextual inquiry and formative Usability testing as crucial for successful Usability design [162]. *Usability Testing* includes task analysis that studies the way people perform tasks with existing systems. By a high-level abstraction study of cognitive processes, we could identify what are the individual tasks that the language is expected to support. For each task we should identify: *Goal*, *Pre-conditions*, *Dependencies*, *User background* and *Sub tasks*.

The cognitive activities that should be analysed in the study are:

1. *Learning* both syntax and semantics;
2. *Composition* of the syntax required to perform a function;
3. *Comprehension* of function syntax composed by someone else;
4. *Debugging* of syntax or semantics written by ourselves or others;
5. *Modification* of a function written by ourselves or others.

Experimenters in human factors developed a list of tasks to capture these particular aspects [168]: *Sentence Writing*, *Sentence Reading*, *Sentence Interpretation*, *Comprehension*,

Memorization, and *Problem Solving*. We can evaluate these tasks using tests such as *Final exams*, *Immediate Comprehension*, *Reviews*, *Productivity*, *Retention*, and *Re-learning*. Performing exhaustive evaluation of different tasks in the language usage is interesting, but would be too expensive. Therefore, the evaluation usually concerns only the most critical activities.

More general method in assessing Usability are *Heuristic evaluation* [155], but are often regarded as not being capable to encompass all Usability attributes. Recently, a new evaluation method called *Domain Specific Inspection* was developed using traditional evaluations in novel ways [162]. Alroobaea [5] proposed a methodological framework which generates a domain-specific evaluation method, which is used to improve the usability assessment process for a product in any chosen domain. This adaptive framework is able to build a formative and summative evaluation method that provides optimal results with regard to the identification of comprehensive Usability problem areas and relevant Usability evaluation method metrics, with minimum input in terms of the cost and time usually spent on employing a Usability evaluation method.

Concerning the **when**, we argued that we should adopt a systematic approach to obtaining a timely frequent Usability feedback, while developing the DSL, to better monitor its impact [17]. This implies the integration of two different and demanding complementary processes: language development and Usability evaluation. Language Engineers should be aware of Usability concerns during development, to minimise rework caused by unforeseen Usability shortcomings. In turn, Expert Evaluators should have enough understanding of the domain-specific models involved in software language development to be able to properly design the evaluation sessions, gather, interpret, and synthesise meaningful results that can support the DSL improvements in a timely way.

The timely frequent Usability testing is in line with agile practices, making them a good fit for this combined DSL building (i.e. software development) and evaluation process (i.e. Usability design) [129]. An agile development process breaks products into small increments, and each iteration should fit in short time-boxes that typically do not last more than a month [59]. This iterative, incremental development process is also in line with Visser's inductive DSL development suggestion [201] (see Sec. 2.1.4). Agile practices provide a method for gathering user feedback by conducting a focus group after a feature was implemented, and ask for users' opinions. However, this approach is insufficient in supporting Usability design which is largely based on observing user behaviour by utilisation of Usability investigation before product implementation [188].

Therefore, it is necessary to apply a **UCD** that is comprised of end user involvement in the development of software products at different points of the life-cycle. UCD includes Usability evaluation techniques that support Usability design such as participatory design, focus group research, surveys, walkthroughs, preliminary prototyping, expert or heuristic evaluation, Usability testing, as well as follow-up studies [157, 172, 207]. UCD can be characterised as a multistage problem-solving process. It foresees how users are

likely to use a product and tests the validity of those assumptions concerning user behaviour in real world tests with actual users. Such testing is necessary as it is often tough for the developers of a product to understand what is intuitive for an end user of their design experiences, and what each user's learning curve may look like. The essential activities required to implement UCD are described in ISO 13407 [38] as:

1. Plan and manage the human centered design process
2. Understand and specify the context of use
3. Specify the stakeholder and organizational requirements
4. Produce design solutions
5. Evaluate designs against requirements

2.2.4 Contextual aspects of DSL Usability evaluation

In this particular research, we only consider languages that are used as communication interfaces between humans and computers (i.e. UIs). Therefore human-human languages, e.g. natural languages, and machine-machine languages, e.g. communication protocols, are not relevant for the work described in this thesis.

Semiotics, the study of the structure and meaning of languages, is a part of linguistics that studies the dependencies and influences among syntax, semantics, and pragmatics. The *syntax* of a language defines what signs we can use in that language, and how we can compose those signs to form sentences. The *semantics* of a language defines the conceptual meaning of the sentences in that language by stating how they can be logically interpreted. Finally, the *pragmatics* sets the context of use from which the sentences of that language can have some logical meaning [179].

The **Context of Use** i.e. *'the users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used'* [103] is one of the characteristics that we must be considered while evaluating DSLs Usability. This is to pragmatically distinguish between the scope of DSLs' assessments. Different DSLs, especially the ones that are developed for various domains, have a different Context of Use. We can infer that the users of those DSLs, most likely, will have different knowledge sets, each one with a minimum amount of ontological concepts required to be able to use each language.

We need a rigorous and collaborative Usability evaluation procedure for DSLs (both during and after the particular DSL development) that supports validation of DSL sentences (called instance models) in a correct Context of Use. According to the context of communication, these sentences can have different interpretations. If the context is not clear, interpretations can be ambiguous. Notice that, in this pragmatic perspective, languages that do not even share the same base syntax of those same sentences may share the same domain concepts, i.e. the intersection of their domain concepts is not empty for

a given non-empty intersection of contexts of use. If the intersection of their contexts of use is empty, then they do not share any of the identified domain concepts.

If we say that *Context of Use* has some ontological purpose, then we can see it as *a problem to be solved* in the language user's mind. One example of this is the set of GPLs where each user has to know about *programming* concepts *(variables*, *cycles*, *clauses*, *components*, *events)*, plus the domain concepts from a given *Context of Use*. Moreover, languages that reduce the use of *computation domain concepts* and focus on the *domain concepts* of the *contexts of use's* problem are called **Domain-Specific Languages**. Notice that, in these pragmatic perspective languages that do not even share the same base syntax may actually share the same *domain concepts*, i.e. the intersection of their *domain concepts* is not empty for a given non-empty intersection of *contexts of use*. If the intersection of their *contexts of use* is empty then they actually do not share any of the identified *domain concepts*.

For example: consider both the *SQL* and *C* languages. The *SQL* language has a reserved word called *table* to represent a database table from a DBMS. There is no *table* in the list of reserved words of *C* language that the user of *C* can immediately read as *table* with the same meaning as read in *SQL* (i.e. a database table from a DBMS). However, one of the *contexts of use* of *SQL* where *table* is applied: *createtable* can be emulated by means of a high level *C* (Application Programmers Interface) API function that have the same purpose of creating a table in the same DBMS. Moreover, if there is no *C* API supported by the DBMS, then we can even imagine how it would be to write it completely in *C* as part of the implementation of *the context of use* stated in *createtable*.

If we perform an analysis of the names of reusable components (in reusable infrastructures), and the reusable data structures and methods from existing APIs, and figure out all the possible ways of how they can be composed in a meaningful way then we can infer a *bottom-up* DSL from that reusable infrastructure. This *bottom-up* method of building languages by reusing existing reusable infrastructures may however generate languages that lack generality in the capability of solving any class of problems of a given domain, or if the domain of the problem is not yet fully bounded (categorized), there may be irregular composition patterns that can be non-sense with respect to. the problem.

A *top-down* method would be to complete the domain analysis phase that is behind the existing reusable infrastructure, by discarding any existing implementation and focusing only on the complete description and categorization of the class of problems from which its users will use our new DSL to describe their solutions while using the identified *problem concepts* with respect to its *context of use*. If we find a mapping between all the possible expressible solutions which might be very difficult in some cases in our new DSL and the existing *concepts* of a reusable infrastructure, then we have assembled a *top-down* DSL.

DSLs are built for a more confined context of use, capturing one particular set of domain concepts. While evaluating these languages, the universe of users is smaller, and they have less diversity of skills, so the validity of the results to their target population is

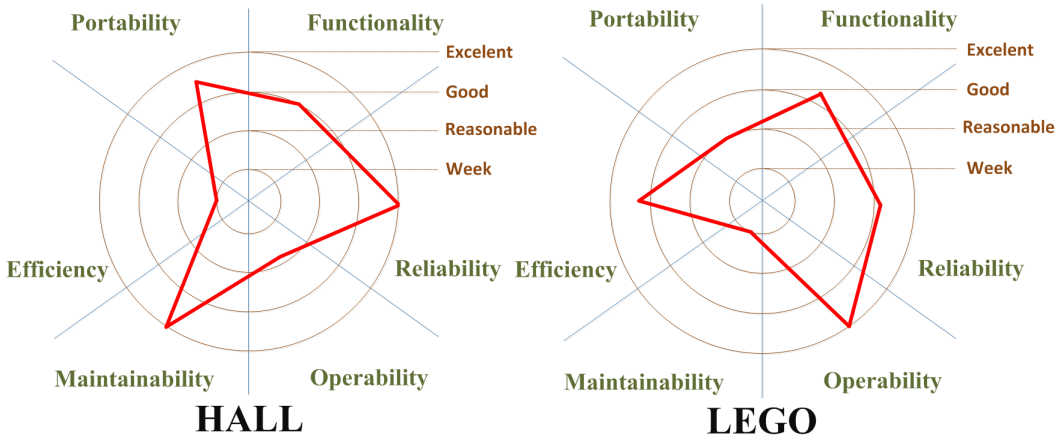


Figure 2.3: Kiviat diagrams for (H)ALL and Lego DSLs (taken from [19])

much higher.

Although typically DSLs have an explicit underlying model (by means of metamodels or grammars), while UIs models are usually implicit in their implementation, in general there is no distinction between using both final products from the user perspective. The influencing characteristics to achieve Quality in Use can be very diverse, namely if we compare two languages in different domains such as the ones presented in Figure 2.3 ((H)ALL [30] and Lego [1] (accessed September 19, 2017)). Here we present an assessment of language non-functional requirements for two languages with different contexts of use. These requirements are presented in terms of internal and external quality attributes from ISO/IEC 9126 [103], for which we predict the expected success level that is necessary for achieving the planned Quality in Use for each of these languages. Different contexts of use lead to different priorities, with respect to quality attributes. For example, we consider external efficiency to be much more important for Lego than for Hall. The waiting time for a language to execute the program on a robot with a Lego can demotivate its users, children, to use it. On another hand, the execution of Hall models is not such a frequent activity of its users and it is acceptable to leave the execution running for certain time, on a cost of investing in high reliability.

Different quality attributes will bring success in the achieved Quality of Use. It depends on its context of use and the target user population, so they can be identified just after performing the Context of Use analysis. Also, it is not always possible to achieve optimal scores for all Usability attributes simultaneously, so, when Usability trade-offs seem inevitable, it is necessary to find a reasonable trade-off solution that can reasonably satisfy both requirements, or, in an extreme case, support an explicit decision concerning which of the conflicting Usability attributes should take precedence.

[1] http://mindstorms.lego.com/en-us/Software/Default.aspx

Chapter 3

Related work

The related studies show that the scientific community is still not making use of the current solutions offered by the software engineering community [113]. The requirements discovery process is difficult because DSLs are exploring new domain-related problems. Therefore, in most of cases requirements cannot be known in advance; they change as knowledge changes [177]. These constraints affect the applicability of many of the traditional software engineering practices and partially explain why scientists tend not to use them [114].

3.1 General-purpose usability evaluation approaches

In the context of GPLs, comparing the productivity impact of using different languages during the software development process has some tradition [165]. Some of the common techniques are the use of a popularity index [60], the cognitive dimensions framework [88], or heuristics based on the studies of cognitive effectiveness for visual syntax [151]. These methods can be reused when these techniques are identified as relevant for the usability of a DSL (e.g. Moody's work on cognitive effectiveness can be reused if a visual concrete syntax is given to the target DSL). When usability problems are identified too late, a common approach to mitigate them is to build a tool support that minimises their effect on the users productivity [35].

GPL users are typically technically-oriented programmers, with good understanding, skills, and experienced in computer science and technology. However, the understanding of domain concepts is also necessary to develop programming solutions. On the other hand, DSLs are meant to reduce the direct use of computation concepts by putting the focus on the domain concepts. As such, DSLs are expected to be used by a target population familiar with the domain concepts, but not necessarily experienced with computer

science and technology (e.g. experts from physics, chemistry, finance, management, etc.). The evaluation criteria for DSLs need to be appropriate for their target users, as well as to their technical, social and physical environment.

We can build on existing methodologies and tools to assess the usability of UIs [19] and adapt them for programming languages (as we discussed in previous Chapter). Existing UI practices, due to the wide spectrum of the context of use that they target, makes it hard to interpret what the collected information means. While for DSLs the population of users tends to be smaller and less diverse. As such the sampled subjects are likely to have a higher proximal similarity to the remaining elements of the population they were sampled for, mitigating the external validity threat.

3.2 The state of practice in DSL evaluation

Some studies address specific quality characteristics for DSLs by performing experimental evaluations after implementing the language. Haugen et al. [97] present a structured questionnaire based on three dimensions of a DSL: expressiveness, transparency, and formalization. Merilinna and Parssinen [146] investigate the benefits of using DSLs by making experimental comparisons between the DSL approach and traditional approaches. Kosar et al. [124] independently evaluated several DSLs and are mostly concerned with program comprehension, correctness and efficiency, while using the DSLs, when compared with using GPLs. A detailed analysis of their data could be used to identify opportunities for improving the tested DSLs. Kieburtz et al.'s [116] experiment addresses DSL comparative evaluation as part of the concern with flexibility. Murray's experiment [152] explicitly looks for opportunities for improving the respective DSLs under scrutiny by taking learnability, understandability, usability, user satisfaction and language evolution as improvement goals. Kärnä et al. [111] evaluate a DSL solution in an industrial case focusing on the productivity and usability. They first determine the objectives for the creation of the DSL and then collect data via controlled laboratory studies. In general, existing assessments are performed with a final version of a DSL when potential problems are expensive to fix. Our research aims to introduce language evaluation concerns early in the DSL development process so that problems can be found 'on-time' and fixed at a fraction of the cost it would take to fix them if detected only after the implementation.

Gabriel et al. [76] present a systematic review emphasizing the reduced concern on the evaluation of DSLs. This work highlights the state of practice and increases awareness to the shortcomings in research regarding this problem. Kosar et al. [122] performed a systematic mapping study of grammar based DSLs covering the period from 2006 till 2013. They concluded that the DSL community focuses more on the development of new techniques/methods rather than investigating the integrations of DSLs with other software engineering processes or measuring the effectiveness of DSL approaches. According to Kosar's study, the primary studies usually discussed the following three DSL development phases: domain analysis, design and implementation, whilst validation

and maintenance have been rarely presented (see Figure 3.1). In many primary studies authors found a brief section on domain analysis identifying the main concepts of DSL under development followed by the design of DSL syntax and semantics and finalizing with implementation details. This study explicitly presents a clear concern regarding the lack of DSL research within the validation phase, in particular controlled experiments. Also, it is pointed that *"DSLs had rarely been validated (e.g., by end-users) assuming that the developed DSLs were perfectly tailored for domains, as well as fitting end-users requirements. However, this is far from true. DSLs under development should be empirically validated, if possible with the collaboration of end users, as well as assessed considering existing research from Psychology of Programming"*.

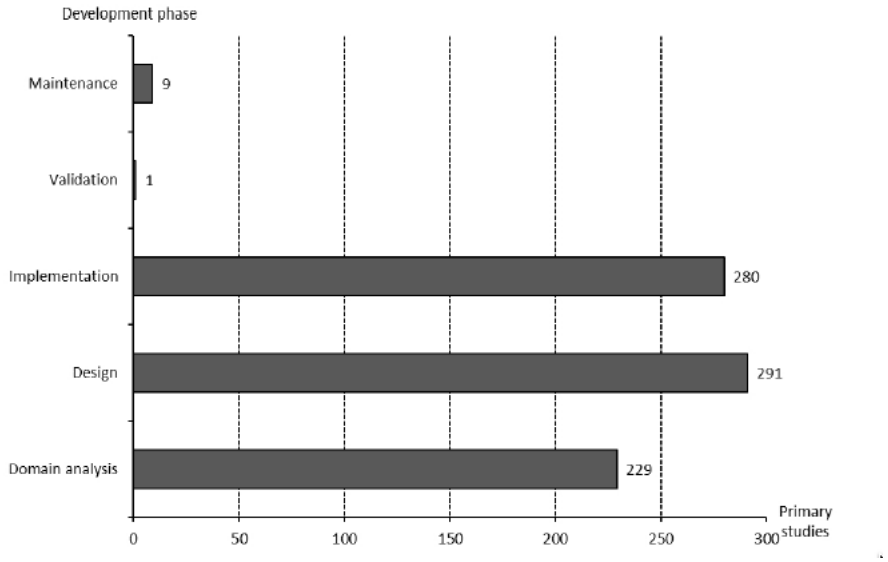


Figure 3.1: Research distribution in DSL development phases (taken from [122])

However, some authors did tackle above mentioned problems. Kolovos et al. [121] list the core quality requirements for a DSL. Hermans et al. [98] identify success factors for designing DSLs by performing an empirical study. Wu et al. [212] present an approach to determine the effort while using DSLs during application development, contributing to the classification of the effort and proposing of related metrics. Kelly and Pohjonen [112] discuss worst practices for creating DSLs which developers should avoid, based on industrial case studies. McKean and Sprinkle [144] present criteria that will help in selecting a DSL or any other approach to be used in system development. According to Nishino [156], if developers do not have the suitable domain knowledge during DSL development, inappropriate abstractions occur as a consequence of conceptual misfits. He proposes to identify usability problems by analysing a set of cognitive dimensions proposed by Green et al. [88]. However, this approach does not indicate how to proceed with applying changes in a potential existing analysis model, how to evolve the design models, or the implementation infrastructure. Further, the approach addresses only partially the testing

of usability problems by usability experts, and it does not include end users in the evaluation. Kahraman [110] proposed a Framework for Qualitative Assessment of DSLs that adapts and integrates the ISO/IEC 25010 standard, maturity level evaluation approach and the scaling approach into a perspective-based model. This framework supports the choice of quality goals from the different stakeholders' perspective, but does not to include the Domain User concerns. Finally, Häser [95] provides an integrated end-to-end tool environment to perform controlled experiments in DSL engineering. The environment supports language design and steps of experimentation, i.e., planning, operation, analysis and interpretation, as well as presentation and package. Controlled experiments have the potential to provide appropriate, data-driven decision support for language engineers to compare different language features with evidence-based feedback. However, this kind of usability assessments is not always cost effective in early phases of language development.

More recent, in 2017, Rodrigues et al. performed systematic literature review related to usability evaluation of DSLs [164]. The authors intended to find out what was the importance of usability considered during the DSL development. Further, they identify what were the evaluation techniques that were applied in the context of DSLs and what were the problems and limitations identified during the DSL usage. They reported the 12 primary studies which address the topic, of which four are authored by the thesis author, namely: Sinha et al. [183] , Barišić et al. [18], Barišić et al. [22], Barišić et al. [23], Rouly et al. [171], Ewais et al. [69], Barišić [20], Gibbs et al. [80], Teruel et al. [190], Kabač et al. [109], Cuenca et al. [62], Albuquerque et al. [2]. Based on the results, authors present DSL usability evaluation taxonomy. Usability evaluation methods which were used are classified as observation methods (A/B test and usability test), inspection methods (heuristic evaluation) and walkthrough. The most of the primary studies were identified to use the experimental study as an empirical method, while the assessments used quantitative and qualitative data types. Regarding evaluation instruments, most of the primary studies used Questionnaire and Interview, while other reported instruments were User Observation, Recording User Action and Heuristic checklists. Two studies used cognitive dimensions framework. Finally, the review identified the following evaluation metrics: Ease of Use, Efficiency, Understanding/ Learning, Effectiveness, Usage Satisfaction, Productivity, Flexibility, Effort/Competition Time, Task Error, Representatives, Error Rate, Perceived Complexity and Intuitiveness. Based on the results of the systematic literature review, authors stressed the necessity for the framework which will support the DSL usability evaluation.

3.3 Applying UCD into design of visual languages

We performed a systematic literature review of the research literature of DSLs development reports in the period of 2009 to 2014 of Journal of Visual Languages in [24]. The quest was to identify if they report domain analysis and/or evaluation of developed

DSL. Based on results of performed analysis we concluded there is increased awareness of usability evaluation in a field of visual DSLs, when compared to results obtained by Gabriel et. al [76]. The majority of the papers did systematically report the realization of some kind of experimental evaluation as well as domain analysis. However, the domain analysis reports don't reveal a systematic approach to its performance, or even the characterization of the solution regarding its technical underneath environment. This makes us ask how much effort will be necessary for integrating DSLs into working environment. Studies are usually reporting a state-of-the-art in a domain, review of good practices, comparisons to previous approach or theoretical analysis. On another hand, most of the languages are developed to be an extension of the previous approach. This is good practice that indicates that actually the conceptual domain models are being partially reused, however, it is rare to find insight into which of these concepts are kept the same and which were changed.

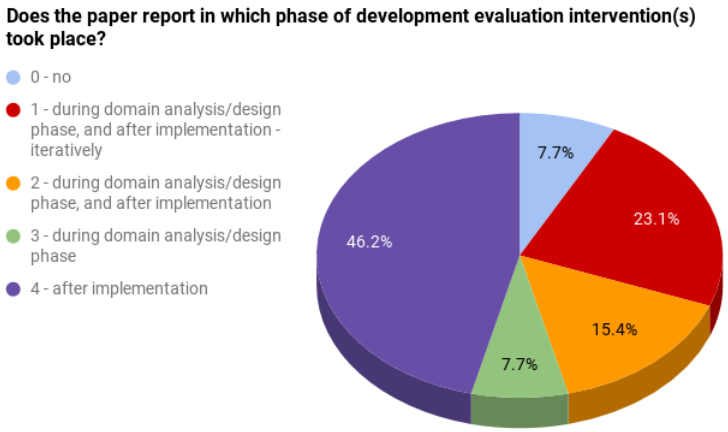


Figure 3.2: Placement of the evaluation assessments into development of visual DSLs (taken from [24])

Concerning the placement of the evaluation assessments into DSL development process, we could get the idea in which phases they took place for the majority of the papers (see Figure 3.2). The practice is still similar as reported by Gabriel et. al, to make evaluations after implementation (over 40%). However, almost 40% of them are introducing evaluation assessments already during the domain analysis or design phase and after implementation, while few of those even claiming to make it iteratively. This early assessments are often introduced by performing heuristic evaluation [33, 78, 81] or using cognitive dimension analysis [42, 91, 153, 176] often involving evaluation experts. On another hand, most of the evaluations are involving the end-users into at least one of the assessments. However, common practice is to perform this with student surrogates for novice users, specially for the assessments of working prototypes or final product. On another hand, the expert users are often used for early assessments.

We highlight the studies which reported explicitly to address the UCD during their development: Neumann [153] investigates rule-based, end-user strategy programming by introducing a domain-specific, end-user programming environment to allow football coaches to create animated football scenarios and by associating strategy information with virtual football players. This study contributes further by putting into evidence the usefulness of Natural Programming design process, which applies principles of UCD [160]. Aghee [1] iteratively evaluated a DSL for developing of Mashups using formative evaluations. Angelini [9] presents an innovative visualization environment, which makes more effective the information retrieval performance evaluation and failure analysis. Environment exploits visual analytic techniques in order to foster interaction and exploration of the experimental data. The environment has been validated through a user study involving experts which showed to eases the interaction with experimental results, supports users along the evaluation process and reduces the user effort. Ardito [10] presents an approach where a composition platform enables the extraction of content from heterogeneous services and its integration. Bauelo [33] presents a visual query system, which allows users to graphically build queries over data streams and traditional relational data. The system has been designed and implemented following the UCD approach. Two different releases of the system have been incrementally and iteratively designed and evaluated. The first release has been evaluated using heuristic evaluation. The second release, whose design was a refinement based on the results of the foregoing heuristic evaluation, was evaluated by several users. Moreover, a comparative evaluation involving users has been conducted on the second release. The approach has been employed in real industrial scenarios, and turn to be beneficial for adoption of product by their users. Danado [64] creates a visual mobile end user development framework which allows end users without programming background to create, modify and execute applications, and provides support for interaction with smart devices, phone functions, and web services. Fogli [71] describes a meta-design approach to transfer the development of government-to-citizen services from professional software developers to administrative employees, without forcing employees to acquire any programming skills.

We find that these examples are valuable, although they lack a systematic experimental assessment in most of the cases. However, evaluation practice found in these works could be made model-driven, systematic and extended to DSLs in general.

3.4 Leveraging domain experts in the DSL development

Finally, we present attempts to solve the usability problems by taking into account end-users needs during DSL development. However, these approaches are in most cases just applicable in the domain analysis and language design phase, namely supporting collaborative development of the abstract and concrete syntax with end-users. However, these examples do not fully represent the complete domain model explicitly (scope, terminology, concepts, and commonalities and visibilities). Also, the profile of the users which

are included, or knowledge sets which are necessary, are essentially domain experts.

The motivation of Perez et al. [66] is to involve end-users in a MDD process. Since end-users do not usually know about DSLs and modelling tools, as professionals do, their goal is to develop a modelling language with a good usability. However, specific requirements and needs of the target end-users are not directly communicated during the DSL development process. In fact, the proposal always creates visual DSLs to improve usability, which may not be the best approach for some end-users. The goal of Wuest et al. [213] was to facilitate the meta-modeling activity to non-experts by creating FlexiSketch, an environment for modelers and end-users to design together the examples of the domain using sketches. These examples are used for the creation of the DSL syntax, both the abstract syntax and the concrete syntax. Similarly, Cho et al. [57], provide a friendly solution for end-users to describe domain examples, the creation of the DSL syntax, and the semantic restrictions. Approach supports the domain examples sketches which are transformed into graph representations and then, an inference engine which obtains the metamodel from those graphs. Kuhrman et al. [127] want to bring together DSL developers and domain experts when developing DSLs in complex application domains. Their work provides a DSL, named 'PDE language', which provides a visualization model editor with different views to facilitate the participation of domain experts in the design of concrete syntax. Sanchez-Cuadrado et al. [174] wanted to support the use of informal drawing tools as a friendly interface that facilitates the meta-modeling task. The difference of this work in respect to the others is the approach used to generate the meta-model from the domain examples sketches. This meta-model is obtained iteratively, one example at a time, in which developers are able to assess the evolution of the meta-model and the specific effects of each domain example over the meta-model. If some changes have been applied, a procedure checks for possible mismatches between the final meta-model and the domain examples. Nevertheless, none of the above approaches makes explicit the domain model details, nor addresses evaluation activities of DSL development.

The motivation of Canovas et al. [49–51] was to highlight the importance of the end-users role in the definition of DSLs and provide means to enable the collaboration between end-users and developers in the context of DSL development. With this aim, they proposed a community-driven DSL Collaboro, to encourage end-user participation in the definition of DSLs. Collaboro describes the elements of the collaborative activity (comment, vote, solution, etc.) and the elements of the abstract and concrete syntax of a DSL (entity, attribute, relationship, textual notation, etc.). These elements are used to track the evolution of both the abstract and concrete syntax. The collaborative development starts after requirements gathering, when developers design a preliminary abstract and concrete syntax. End-users are able to comment, propose solutions, and vote other participants opinions and proposals. This interaction continues until the syntax, after all the changes are proposed and applied. In summary, the main contribution of this work as a whole, in contrast of previous works, is the use of friendly mechanisms, such as the use of an informal panel and the use of examples, as a way to reason about the domain

and the abstract and concrete syntax. Villanova [200] made a significant contribution in introducing agile methodologies in DSL development. Methods like customer reviews and demonstrations that are used in Scrum are not often validated by real end-users, but mostly with managers or 'buyers' of the software. The practice of introducing a definition of *Done* brought by the agile practices promotes demonstrations to the customers and a feedback collection. However, demonstrations and questionnaires are not sufficient to test product usability with the customers. Indeed, without using a product, end users will have a hard time identifying where they are likely to make mistakes and use the product inappropriately. While they can provide some feedback on their satisfaction regarding language construction (as far as they can observe it from the demonstration), they will not be able to provide feedback on their efficiency and effectiveness while using it.

The mentioned works do not address how to prioritize user recommendations, for the next iterations of the DSL development. We could leverage user profile to support prioritization of requirements for the next iterations, which would lead to more timely and usable improvements in the DSL. We need to be aware that the users while using software products are often not aware of their mistakes and inappropriate use of the product. Methods which support collection of a user feedback regarding their satisfaction with provided interface elements or functionalities, are valuable in the beginning of the development cycle. However, later on it is necessary to obtain as well proofs about their efficiency and effectiveness while solving a domain related problems. The experimental evaluations with end users are expected to uncover the set of the mostly needed functionalities in the application area that are possible to be developed in the following cycle when considering the technical restrictions (e.g. lack of informational database to support the functionality, need to develop previous modules in order to support the functionality). The domains for which DSLs are developed are constantly changing, so prioritizing evaluation can help in providing timely solutions that are usable to their target end-users. However, the specific requirements and needs of the target end-users need to be evaluated by including them in systematic evaluations during the DSL development process.

Chapter 4

Usability evaluation approach for DSLs

Evaluation with users, known as Empirical Evaluation, is recommended at all stages of development, if possible, or at least in the final stage of development [154]. Each type of measure is usually regarded as a separate factor with a relative importance that depends on the Context of Use. Iterative testing with small numbers of participants is usually preferable, starting early in the design and development process.

4.1 Experimental model

We argue that the quality in use of a DSL should be assessed experimentally. In Software Engineering, a controlled experiment can be defined as *'a randomized experiment or quasi-experiment in which individuals or teams (the experimental units) conduct one or more Software Engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages or tools (the treatments)'* [184]. For our purposes, this can be instantiated with developers typically conducting software construction, or evolution tasks, for the sake of comparing different languages – including the DSL under evaluation and any existing baseline alternatives to that DSL.

4.1.1 Experiment activity model

Figure 4.1 outlines the activities needed to perform an experimental evaluation of a software engineering claim, following the scientific method. During *requirements definition*, the problem statement (i.e. research questions), experimental objectives and context are defined. The next step is to perform *design planning*, where context parameters and hypotheses are refined, subjects are identified, a grouping strategy for subjects is selected, and a sequence and synchronization of observations and treatments for each of the experimental groups is planned. The sequencing and synchronization of such interventions,

their nature (observations or treatments) and the group definition policy, define the *experimental design*. The data collection activities plan is also set during design planning. This is followed with *data collection*, which often includes a pilot session, to correct any remaining issues, and the evaluation itself, following the designed plan. This step is followed by *data analysis* where data is described in the form of statistical tables and graphs, and, if necessary, the data set is reduced. Hypotheses are then tested. Finally, during *results packaging*, the results are interpreted and possible validity threats and lessons learned are identified. A detailed discussion on how this process can be followed in a software engineering experimentation context can be found in [83, 84]. Experimental reporting guidelines, generally followed by the experimental software engineering community, are also available [105]. By reporting a given language's quality in use, and the evaluations adhering to such guidelines, the overall ability to make study replications (for independent validation and validity threats mitigation) and its meta-analysis (for building a body of knowledge supported by the evidence collected in different contexts) is expected to increase.

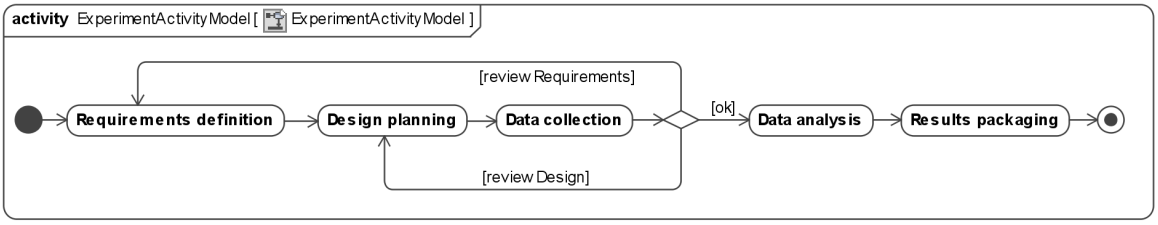


Figure 4.1: Experiment Activity Model Overview (taken from [20])

4.1.2 Experiment design model

In order to contrast the selected DSL experimental validations, we start by modelling their relevant information. This is captured in the class diagrams, adapted and extended from [83]. In a nutshell, this model partially captures some of the essential information of an experimental language evaluation, namely the details on evaluation requirements and planning.

4.1.2.1 Problem Statement design model

Before conducting an experimental language evaluation, one should start by clearly defining the problem that the evaluation will address as modelled in Figure 4.2. This includes identifying where this problem can be observed (i.e., its context, typically where the language will be used), and by whom (i.e., the stakeholder who is affected by the problem – e.g., the language user). It is also important to state how solving the identified problem is expected to impact on those who observe it, and which quality attributes will be affected. The class *QualityAttribute* can take values that are defined in Quality model from ISO Standards (see Figure 2.2).

When conducting language evaluation experiments, one should clearly define the experiments' objectives. Building upon Basili's earlier work [31], Wohlin et. al. proposed a framework to guide the experiment definition [211]. The framework is to be mapped into a template with the following elements: the object of study under analysis, the purpose of the experiment, its quality focus, the perspective from which the experiment results are being interpreted, and the context under which the experiment is run.

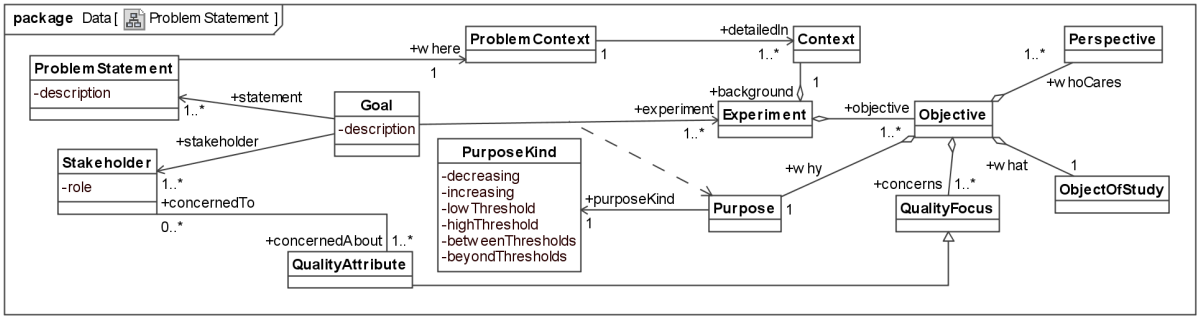


Figure 4.2: Problem Statement design model (taken from [20])

While the experiment definition expresses something about why a particular language evaluation was performed, the experiment planning expresses something about how it will be performed. Before starting the experiment, decisions have to be made concerning the context of the experiment, the hypotheses under study, the set of independent and dependent variables that will be used to evaluate the hypotheses, the selection of subjects participating in the experiment, the experiment's design and instrumentation, and also an evaluation of the experiment's validity. Only after all these details are sorted out should the experiment be performed. The outcome of planning is the experimental language evaluation design, which should encompass enough details in order to be independently replicable.

A example of instantiation of Problem Statement model is given in Figure B.18.

4.1.2.2 Context design model

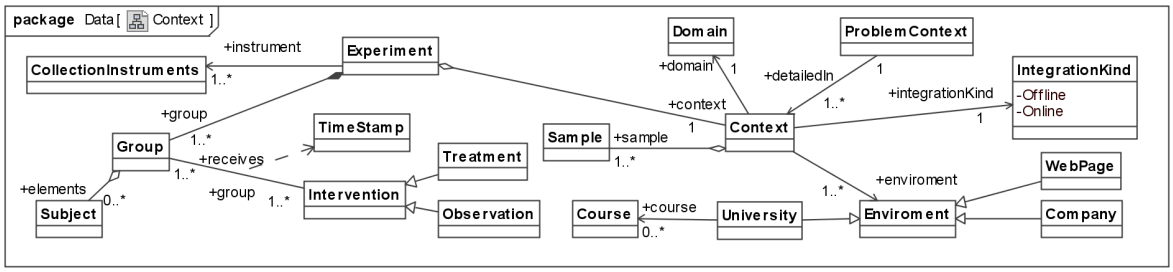


Figure 4.3: Context design model (taken from [20])

Figure 4.3 includes information on the context, including where the experimental language evaluation will take place. The context of an experiment determines our ability

to generalize from the experimental results to a wider context. Experiments can be conducted in different contexts, each of them with their own benefits, costs, and risks. These constraints have to be made explicit, in order to ensure the comparability among different studies and to allow practitioners to evaluate the extent to which the results obtained in a study, or set of studies, are applicable to their own particular needs. Throughout the experiment, there are a number of context parameters that remain stable and their value is the same for all the subjects in the experiment during the whole process. Therefore, we can safely assume that differences observed in the results cannot be attributed to these parameters, while the actual parameters to be reported may vary [211]. Concerning their integration within the language development process, experiments can be conducted either *online*, or *offline*. The former, carried as part of the software process in a professional environment, involves an element of risk, since experiments may become intrusive in the underlying development activity. This intrusiveness may even manifest itself through resources and time overheads on a real project. A common alternative is to carry out the experiment offline.

An experimental language evaluation design prescribes the division of our sample into a set of groups, according to a given strategy. Each of those groups receives a set of **interventions**, which may be either **observations** where data is collected or **treatments**, where the groups receive some sort of input (e.g., training in using a language). The association class with the time stamp allows this data to be ordered in time so that a sequence of observations and treatments can be established. The sequencing and synchronization of such interventions, their nature, and the group definition policy, define the **experimental design**.

A example of instantiation of Context model is given in Figure B.19.

4.1.2.3 Instrument design model

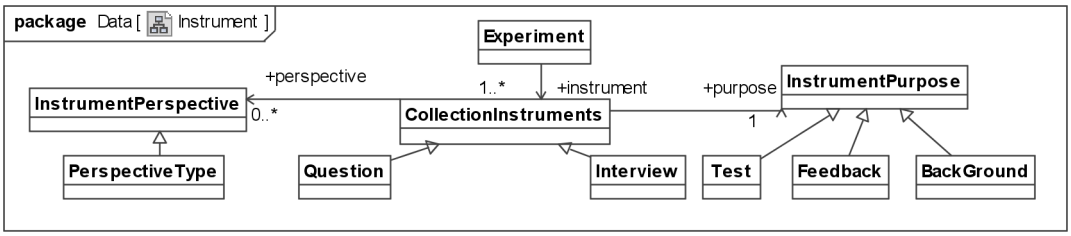


Figure 4.4: Instrument design model (taken from [20])

The **instrument design** presented in Figure 4.4 includes the definition of the artifacts that will be used in the experiment. For instance, in a language evaluation experiment, the syntactical problem instantiation specified with a language can be used as an artefact that will then be changed by the evaluation participants. These changes could be monitored, using collection instruments such as those depicted in Figure 4.4 – e.g., a combination of a test with a post-test questionnaire. This kind of evaluation allows addressing the

instrument perspectives as cognitive activities that are fundamental to assessing the usability of a language, and the quality of instantiation, especially during modification (see, for instance, the usage of cognitive dimensions in [124]). The instrumentation also concerns the production of guidelines and tools (not necessarily computer-based ones) that will support the measurements performed in the experiment. The rationale is to foster the comparability of the collected data by streamlining data collection in a consistent way. Note that instrumentation may also include any training material distributed to the participants, before their participation in the experiment.

A example of instantiation of Instrument model is given in Figure B.20.

4.1.2.4 Sample design model

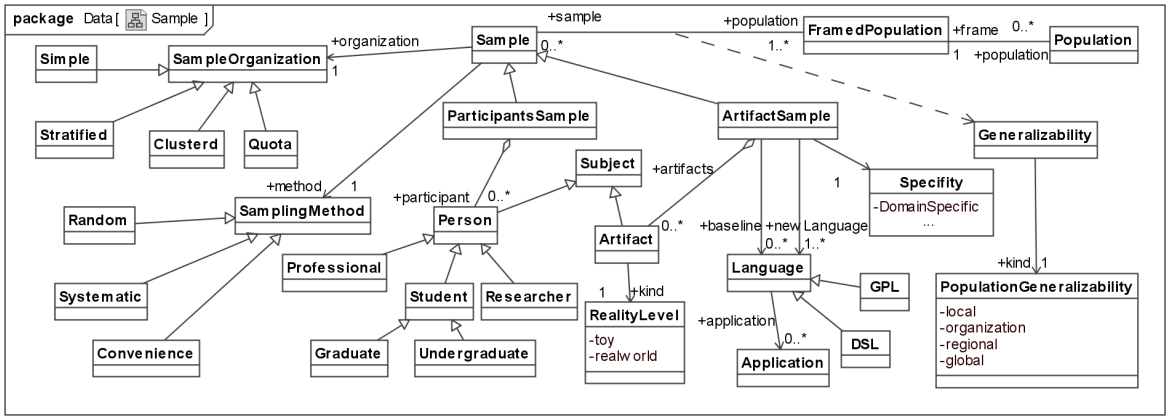


Figure 4.5: Sample design model (taken from [20])

In Figure 4.5, we can see the **sample design** model that includes the participants' profile and the artefacts used in the language evaluation. An orthogonal classification of context concerns the people involved in the language evaluation. One may choose among performing the language evaluation with professional practitioners, or with surrogates for those practitioners (e.g., students). The first option leads to results that are more easily comparable to others obtained in a professional context, but care must be taken to reduce potential overheads to practitioners' activities. Using students as surrogates for professional practitioners is less expensive, but makes the experimental results harder to extrapolate for a professional community. In order to reduce this gap between the students and the practitioners, the researcher should prefer using graduate students, whose expertise is closer to that of novice practitioners.

It is common to use a frame of the population, if it is not feasible to identify all the population's members. In contrast, all members of the chosen population frame are identified. For example, rather than considering all the language components available, one can use a frame that considers only the selected language components as the population. Often, it is not possible to perform the evaluation using all the relevant framed population as evolution subjects. Instead, a sample of that framed population is chosen using

a selected sampling technique, with the objective of being as much representative of the framed population as possible, considering the available resources of the experimenter.

Yet another dimension constraining the language evaluation is the usage of toy vs. real problems. There are at least two issues that motivate the usage of toy problems: the resources available for the language evaluation and the risks concerned with the outcome of the evaluation. The former results from the often very limited amount of time that the subjects can devote to the evaluation. The latter relates to the potential harm caused by the outcome of the evaluation (e.g. while experimenting with using different languages on a real problem, a language that leads to worse productivity can lead to additional costs to a customer). The question, here, is whether the results obtained with a toy problem will scale up to real problems, or not. Toy problems are often used in early evaluations, as their usage is less expensive. If the results of evaluations conducted with toy examples are satisfactory, the risk of scaling up the problem to a real one may be mitigated to a certain extent, although it will not be completely eradicated.

The artefacts used in these evaluations can be generic or domain-specific. When comparing programming languages it is common for these artefacts to be domain-specific, regardless of the original language they were built with. This means, that we can use this model, taking into consideration this attribute specification, to compare GPLs, DSLs, or GPLs vs. DSLs.

A example of instantiation of Context model is given in Figure B.21.

4.1.2.5 Hypothesis and Variables design model

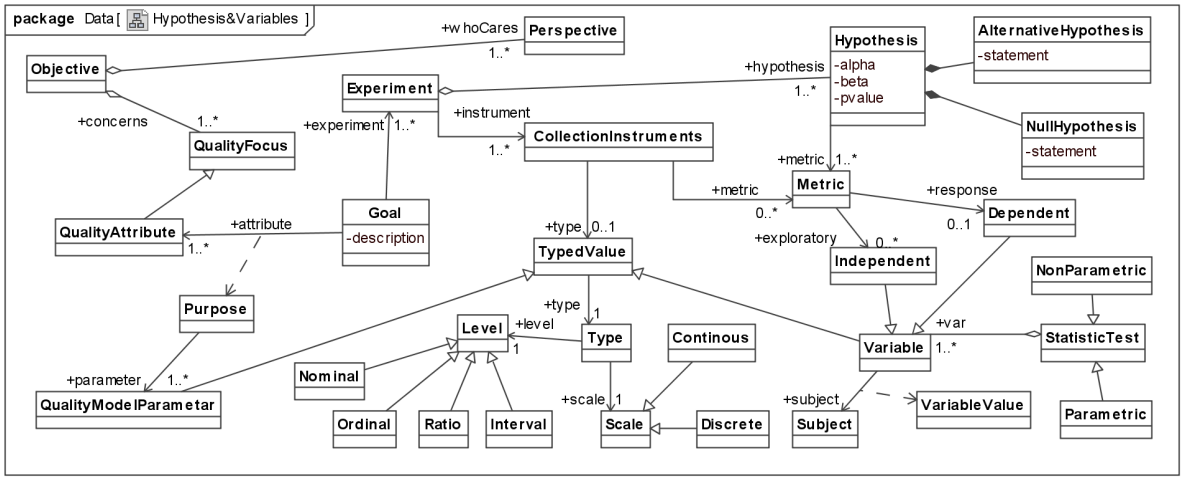


Figure 4.6: Hypothesis and Variables design model (taken from [20])

Figure 4.6 includes the hypotheses tested and the variables used with their characteristics, such as type, scale, and level. The hypothesis formulation should be stated as clearly as possible and presented in the context of the theoretical background it is derived from. The *null hypothesis* states that there is no observable pattern in the experimental

evaluation setting, so any variations found are resulting from coincidence. This is the hypothesis that the researcher is trying to reject. The alternative is that the variations observed are not resulting from coincidence. When the null hypothesis is rejected, we can conclude that the null hypothesis is false. However, if we cannot reject the null hypothesis, we can only say that there is no statistical evidence to reject it. Conversely, if we reject the null hypothesis, we can accept its alternative. If we cannot reject the null hypothesis, we cannot accept the alternative.

Hypothesis testing always assumes a given level of significance denoted by alpha, which represents a fixed probability of wrongly rejecting the null hypothesis, if it is in fact true. The probability value (p-value) of a statistical hypothesis test is the probability of getting a value of the test statistic as extreme as or more extreme than that observed by chance alone, if the null hypothesis is true. Figure 6 presents the relationships between the main concepts involved in hypotheses definitions, starting from the overall objectives of the research, through the specific goals of the experiment, and the questions that will allow assessing the achievement of the goals. The hypotheses are then assessed using metrics.

The language evaluator selects both dependent and independent variables. Dependent variables should be explicitly tied to the research goals (in the context of this chapter, these typically involve evaluating DSLs), and chosen for their relevance with respect to those goals. When it is not feasible to collect direct measures of the level of achievement of the research goals, surrogates can be used, although such replacement is to be avoided, when possible, and clearly justified. When not – e.g. when assessing the usability of a DSL – we may use effectiveness in specifying a system with it as a surrogate for the DSL's usability. Similarly, independent variables are chosen according to their relevance to the research goals.

The analysis techniques chosen for the language evaluation experiment depend on the adopted language evaluation design, the variables defined earlier, and the research hypotheses being tested. More than one technique may be assigned to each of the research hypotheses, if necessary so that the analysis results can be cross-checked later. Furthermore, each of the hypotheses may be analyzed with a different technique. This may be required if the set of variables involved in that hypothesis differs from the set being used in the other hypotheses under test. Discussions relating statistical tests (in particular, parametric vs. non-parametric ones) with variable types can be found in statistics text books, such as [140].

A example of instantiation of Hypothesis and Variables model is given in Figure B.23.

By capturing a rich set of data of a language evaluation, we can pave the way for further analysis, where the information collected in several independently conducted language evaluations can be combined. To do so, the next step is to instantiate this model. In Figure 4.7, we illustrate a partial instantiation of this model, using information collected from the family of language evaluation experiments described [125]. This particular example is chosen for illustration because that family of evaluation experiments is an

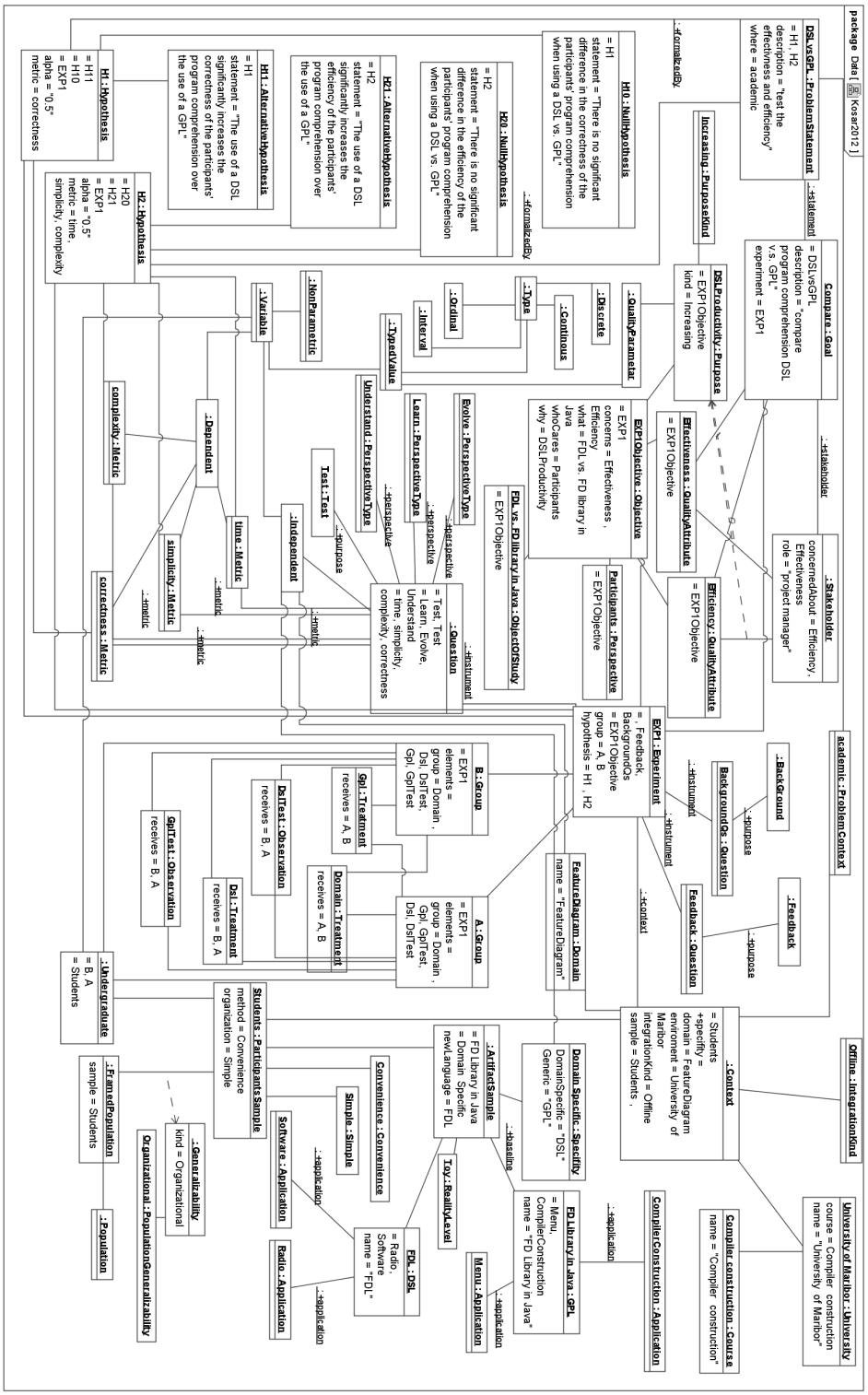


Figure 4.7: Experiment design model instantiation, from info in [125] (taken from [20])

excellent example of how DSL properties validation can be performed in a sound way. The instantiation is only partial, as the whole instantiation would be extremely cluttered.

4.1.3 Experiments overview

The main point in streamlining the evaluation of DSLs and making information available in a common framework is that we can build upon that framework an evidence-based body of knowledge on DSLs and their properties with respect to their usability. We performed a systematic comparison of four language evaluation experiments. They were examples of best practices in languages evaluation with a concern on usability, from which we could perform some meta-analysis, leading not only to a collection of lessons learned "from the trenches", but also to the identification of opportunities to further improve existing validation efforts.

The selected studies are Kieburtz [116], Murray [152], Kosar [125] and Barišić [18] (Section 9.1), which are reported in Table 4.1. The first column represents a specific criterion that we will use in our comparative overview of these studies. The four remaining columns provide information on each of the selected studies. Kosar et al. conducted a family of three experiments, while the remaining selected studies are single experiments. The generic lack of families of experiments, rather than single experiments is a long identified shortcoming in the experimental validation of software engineering claims, so this should be highlighted as a very strong point in this work. Families of experiments help mitigating validity threats that occur in single experiments. In this particular case, the fact that the tested hypotheses have consistent results in all the three experiments in the experiment family increases the confidence in the soundness of the obtained results. Ideally, there should also be experiments within the family run by completely separate research groups, so that any biases by the experiment team that might exist would also be removed. Independent replication of experiments is a standard practice in other domains. For example, the Cochrane Collaboration [1] supports a common repository for health care evidence, which is fed by independently run families of experiments.

Back in 1997, Brooks advocated that meta-analysis should be used to combine the results of independent study replications in Software Engineering [45]. Miller attempted to perform a meta-analysis on a set of independent defect detection experiments, but found serious difficulties concerning the diversity of the experiments and heterogeneity of their data sets, and was unable to derive a consistent view on the overall results [149]. A noticeable feature in the quality concerns row is that, either directly or indirectly, all these studies are concerned with the quality in use of a DSL, including perspectives such as its effect on the productivity of practitioners, which is sometimes indirectly assessed through the effectiveness and efficiency of the language usage. This is, of course, not surprising, as these examples were chosen precisely because they illustrate how such

[1] http://www.cochrane.org/ (accessed September 19, 2017)

Table 4.1: Experiments overview

Criteria	Kieburzt1996 [116]	Murray1998 [152]	Kosar2012 [125]	Barišić2011 [18]
Experiment runs	Single	Single	Family of 3 runs	Single
Quality concerns	Flexibility, productivity, reliability, usability	Learnability, understandability, usability, user satisfaction and language evolution	Effectiveness, time frame, efficiency, usability, perceived complexity	Effectiveness, efficiency, self-confidence in results and language evolution
Context	In-vitro, offline	In-vitro, offline	In-vitro, offline	In-vitro, offline
Comparison	DSL vs. GPL	Visual DSL vs. Textual DSL	DSL vs. GPL	DSL vs.GPL
Participants profile	Professionals	Graduate students	Graduate students	Graduate students
DSL	MTV-G	Kaleidoquery	FDL, DOT, XAML	Pheasant
Baseline	ADA templates	OQL (textual DSL)	FD library in Java, GD library in C, Windows form Library in C#	BEE/C++
Materials origin	Industry-level	Academic	Academic	Academic
Training in DSL	Yes	Yes	Yes	Yes
Training in Baseline	Yes	Yes	Yes	For inexperienced users
Group participants	2 similar groups	4 similar groups	6 similar groups	4 similar groups
Independent variables	Language type, participant	Language type, language factor, experience	Language type,domain, question type, experience	Language type, question type, experience
Dependent variables	Effort, effort/task, acceptance test failures, task difficulty classification, and perceptions on flexibility, productivity and confidence.	Correctness, user preferences concerning both languages	Program comprehension, time, efficiency, simplicity of use, test complexity	Time, Correctness, Confidence scale
Analysis	ANOVA	Paired sample T-test, independent samples T-Test	Wilcoxon Signed Ranks Test	Wilcoxon Signed Ranks Test, Sign Test

evaluation can be performed, in different contexts. Kosar et al.'s work [125] is an independent evaluation of several DSLs and is mostly concerned with program comprehension correctness and efficiency while using the DSLs when compared with using GPLs. A detailed analysis of their data could be used to identify opportunities for improving the tested DSLs. Kieburtz et al.'s experiment [116] addresses DSL evolution as part of the concern with flexibility. The remaining two experiments explicitly look for opportunities for improving the respective DSLs under scrutiny.

The four studies are run in vitro (i.e., in the laboratory, under controlled conditions), off-line. This context is particularly interesting in that the researchers can better control extraneous factors that would otherwise bring validity threats to each of the experiment. Being offline, the risks for the organizations where the studies are conducted are also mitigated, in the sense that if anything goes wrong with the experimentation, this will have no visible effect to external stakeholders (e.g., clients that were considering using a DSL). The downside for this is that there are validity threats concerning the realism of an assessment performed in vitro, as well as that of conducting the experiment offline. Clearly, there are interesting research opportunities to mitigate these threats, by evaluating the same DSLs in a real-world, uncontrolled environment, to strengthen the external validity of the obtained results. The same holds for selection of participants in the experiment, where, whenever possible, real users of the DSL should be involved.

The number of participants is also an issue, due to the relatively high costs of engaging real users in the validation of languages. Concerning this, we would highlight Kieburtz's experiment [116] as it shows how a meaningful assessment can be performed, even with a very low number of participants (only 4). Of course, for statistical soundness, larger numbers of subjects should be used, but, as noted by usability experts, a small number of users can still detect a high number of usability improvement opportunities in a product [155]. Using a small number of participants is an interesting option in early evaluations aimed at identifying defects of the language, to reduce costs. In order to draw more definitive conclusions (with high reliability and validity) that state if the language is better than the previous baseline it is necessary to use a larger number of participants. For instance, in Kieburtz's experiment, the conclusions were sound with respect to the participants, but had a threat with respect to their external validity: with only 4 participants, it was not possible to rule out the possibility of their individual skills playing a role in how the competing languages were evaluated. A similar comment might be made for the evaluation experiments described by Murray [152] and Barišić [18], with 10 and 15 participants, respectively. In isolation, each of these experiments has its own external validity threats. Interestingly, if we combine the results in all these experiments, a consistent pattern of DSL success starts to emerge. Last, but not the least, several of these evaluation experiments uses academic examples for validation, rather than 'real-world' problems. This is, of course, a convenience constraint which entails the obvious threat of external validity, if the examples are not representative of the actual tasks real users will have to perform with the DSLs. Even with real-world examples, the (lack of) coverage of

the DSL language with those examples is also a common threat.

In all these DSLs, there is a high variability of domains and techniques to build DSLs, suggesting that the lessons learned from this collection of language evaluation experiments should, in principle, apply to DSLs from other domains. All the selected studies compare DSLs with an existing baseline which is, in most cases, a GPL-based solution. The noticeable exception is Murray's experiment, where a graphical DSL is contrasted with the textual notation it is built upon. These examples also illustrate how, in most reported cases, the usability evaluation of DSLs is performed once. In a UCD process, this should not be the case. As such, we would expect to find DSL usability assessments covering several versions of the same language, thus supporting the language evolution. Language evolution is covered in some of these studies, usually in the final questionnaire that is prepared for participants, in the end of the evaluation. This feedback can be valuable for language engineers, but the effect of implementing the changes suggested by participants' feedback should ideally also be assessed by a new replica of the experiment, to run with the new version of the DSL, like we did in case of Visualino (Section 9.4, Annex IV).

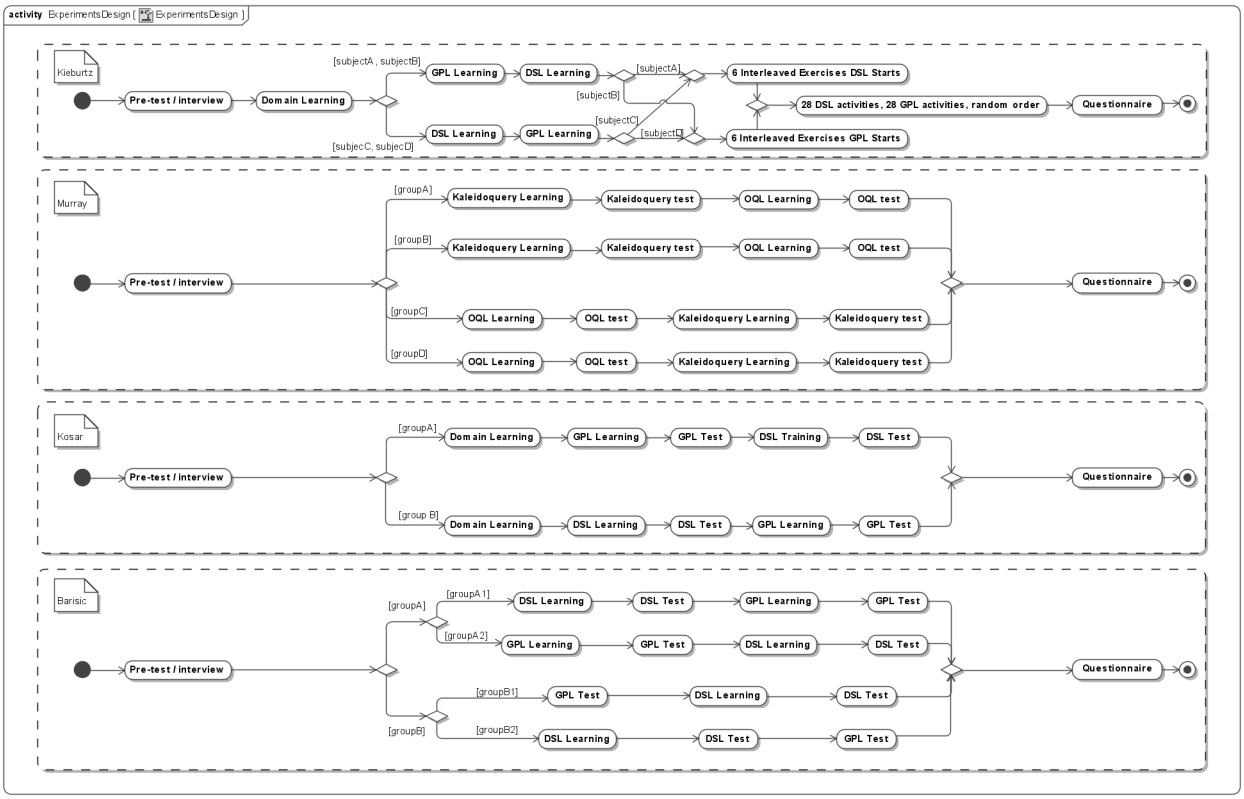


Figure 4.8: Experiments design: Observations and Treatments (taken from [20])

Concerning the experimental designs (see Figure 4.8), whether implicitly or explicitly, they all report collecting some background information. In some of them, domain training was necessary, while in others, it was not. One of the common concerns in all

experiments was to cancel possible learning effects, by splitting participants into at least a couple of groups, so that one of the groups would learn the baseline first and then the DSL, while the other group would have its training and testing path in the reverse order. Whenever more than one category of participants existed (e.g., programmers vs. non-programmers), the groups were further split so that there was a balanced number of experienced and non-experienced subjects following each of the training and testing paths. Experiments usually ended with a questionnaire, so that participant's perceptions on their performance in the experiment, as well as suggestions for improvement in the languages, or other relevant information could be recorded.

All experiments used some statistical approach to assess the extent to which the differences in collected data between using the DSL, or the baseline, were significant. In all cases, some statistically significant differences in the results were reported. These differences should be regarded as indicators of a tendency, rather than as definitive, due to the already discussed external validity issues of these experiments, when considered in isolation, but their overall consistency gives us some trust on the observed trends. In all experiments, the quality impacts of using DSLs vs. using the existing baselines are noticeable, and strengthen the claims concerning a stronger usability using DSLs, when compared to their baselines, with an impact on the productivity of professionals using them, in these tests. We also note how, whenever there is a separation among experienced and non-experienced test participants, the improvement effects are more noticeable in the non-experienced participants. The overall feedback, usually collected through a mix of Likert-scale questionnaires (e.g., each answer is encoded in a symmetric scale expressing the level of agreement with a given statement, ranging from a strong agreement to a strong disagreement), and open questions is, in general, favorable to DSLs, or indifferent, but only rarely favorable to the baseline.

The obvious conclusion of all these studies is that, in general, the analyzed DSLs outperformed their baselines, confirming the anecdotal stories on the benefits of DSLs, with varying differences between the baselines and the DSLs. This is not surprising for at least two motives: (i) those DSLs were built to be a better alternative than the baselines they were compared with, in most cases, so the language engineers had a grasp of how to improve on the existing baselines – the DSLs were built to be good at those tasks they were tested with so, the tests showed that this objective was met; (ii) taking a skeptic's view, it is also arguable that, due to publication bias, we are mostly bound to have assess to success stories, rather than failure ones. A proponent of a new language is less likely to write a report explaining how the language fails to meet some of its goals, whereas the author of a successful language is interested in illustrating, through validation, the advantages of using the new language. This skeptic's view is a strong argument for the independent validation of claims on DSLs' advantages over existing baselines. That said, it should be noted that Kosar's family of validation experiments is an independent one, in the sense that the evaluators are not simultaneously the developers of the solutions under comparison.

4.2 Iterative User-Centered Design approach

We claim that the usability of a language needs to be evaluated by involving the language's end users into development. To be able to identify potential quality problems that will lead to user interaction and experience problems, a suitable approach is to apply UCD practices during design and development of the language. Nevertheless, it is found hard to control budget and plan time and responsibilities accordingly. An incremental, iterative process should be applied to enable tracking of design changes and validation of usability metrics.

4.2.1 Process for performing usability evaluation on DSLs

In order to propose a process for performing usability evaluation on DSLs, we first must ask what are the main goals of a language engineer when devising a new DSL. The main design objectives for building a new DSL are:

- To build a comprehensive language that captures domain expressivity.
- To achieve compliance with existing standards in a given domain.
- To overcome previously identified problems in the domain.

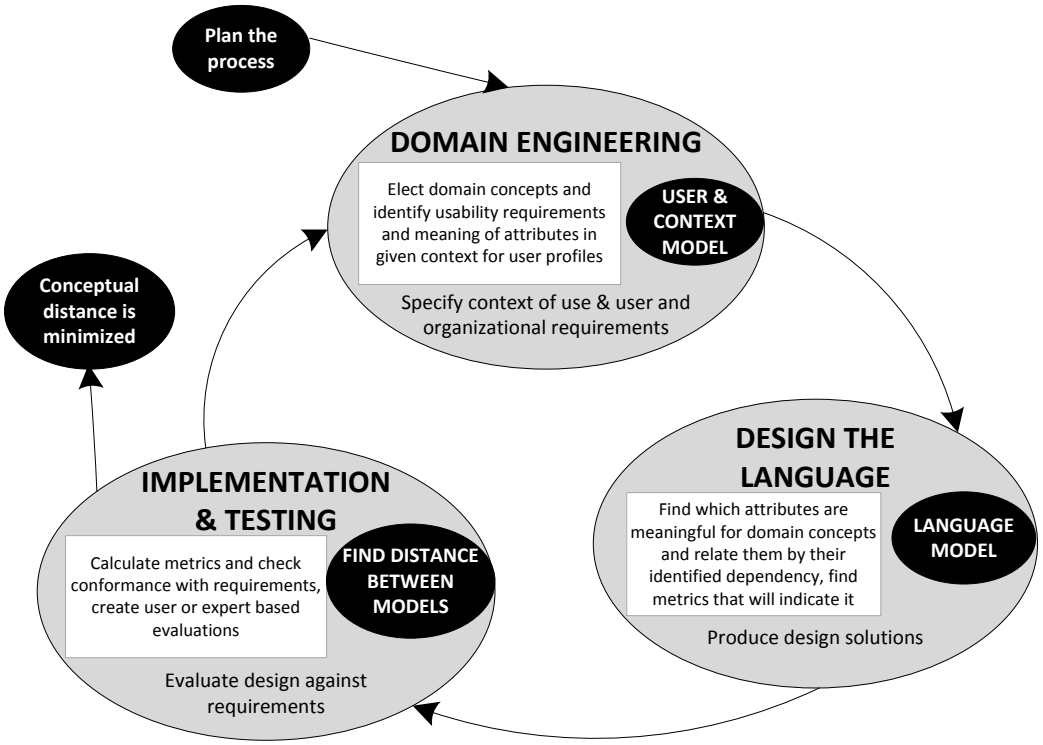


Figure 4.9: Evaluation Process for DSLs' Usability (taken from [17])

As in other usability evaluation methodologies, Usability evaluation should be embedded in the DSL development process (Section 2.1.4), and considered from the beginning of its development together with UCD activities from *Section* 2.2.3 as we propose in *Figure* 4.9. According to our proposal, Usability requirements should be identified during *domain analysis phase* of DSL's construction i.e. while eliciting domain concepts. A first step would be to understand and specify the Context of Use of DSLs. In order to achieve that, it is necessary to engage interviews or questionnaires with the DSL's intended end users in order to capture information about their working environment and the products that are already used within the domain. It is necessary to identify characteristics that the users find useful, frustrating or lacking while using the existing approach to solve the problem and group them in the usability requirements. A set of evaluation goals can be identified during the requirements' elicitation process.

In the *language design phase*, it is necessary to validate if the design decisions conform to the context of use (e.g. chosen technical environment is integrable with the users' environment, or that the designed DSL constructs are understandable). Unclear assumptions can be improved through different iterations, but already with the first designs, the Expert Evaluator can define questions that could answer evaluation goals and which quality attributes from *Figure* 2.2 are meaningful for the implemented domain model. For each domain concept, it is necessary to identify or predict both its frequency and relevance within the domain.

During the *implementation phase*, the evaluation model can be specified. The stakeholders that should be involved in the evaluation could already be profiled and prepared for the execution. In certain cases, just use of proactive approach can be sufficient to evaluate the iteration objectives.

Finally, in the *evaluation phase*, the Expert Evaluator executes experiments and analyses the results, while the Language Engineers may still perform tests or prepare the product for Deployment. Preferably, during the evaluation the users are given real problems to solve in order to cover the most important tasks identified in the domain. Data about satisfaction and cognitive workload should also be evaluated subjectively through questionnaires. It is especially important in this phase to measure all the learnability issues, since DSLs should be (in principle) easy to learn and remember. Of course, in order to certify that we are creating a good DSL we should conduct a comparative analysis with previous products that are already used in the domain and also were built to achieve the same goals.

The main idea is that we can measure the distance between the language domain model and the usability-context model during language development through defined quality metrics that influence Usability. The smaller the conceptual distance, the higher the level of achieved Quality in Use.

The proposed iterative UCD evaluation approach can be merged into the development cycles of already existing and evolving languages. It enables us to intervene at any point of the DSL development. Very often the developers only become concerned with Usability

issues in later phases of the DSL life-cycle.

4.2.2 Pattern language for DSL usability evaluation

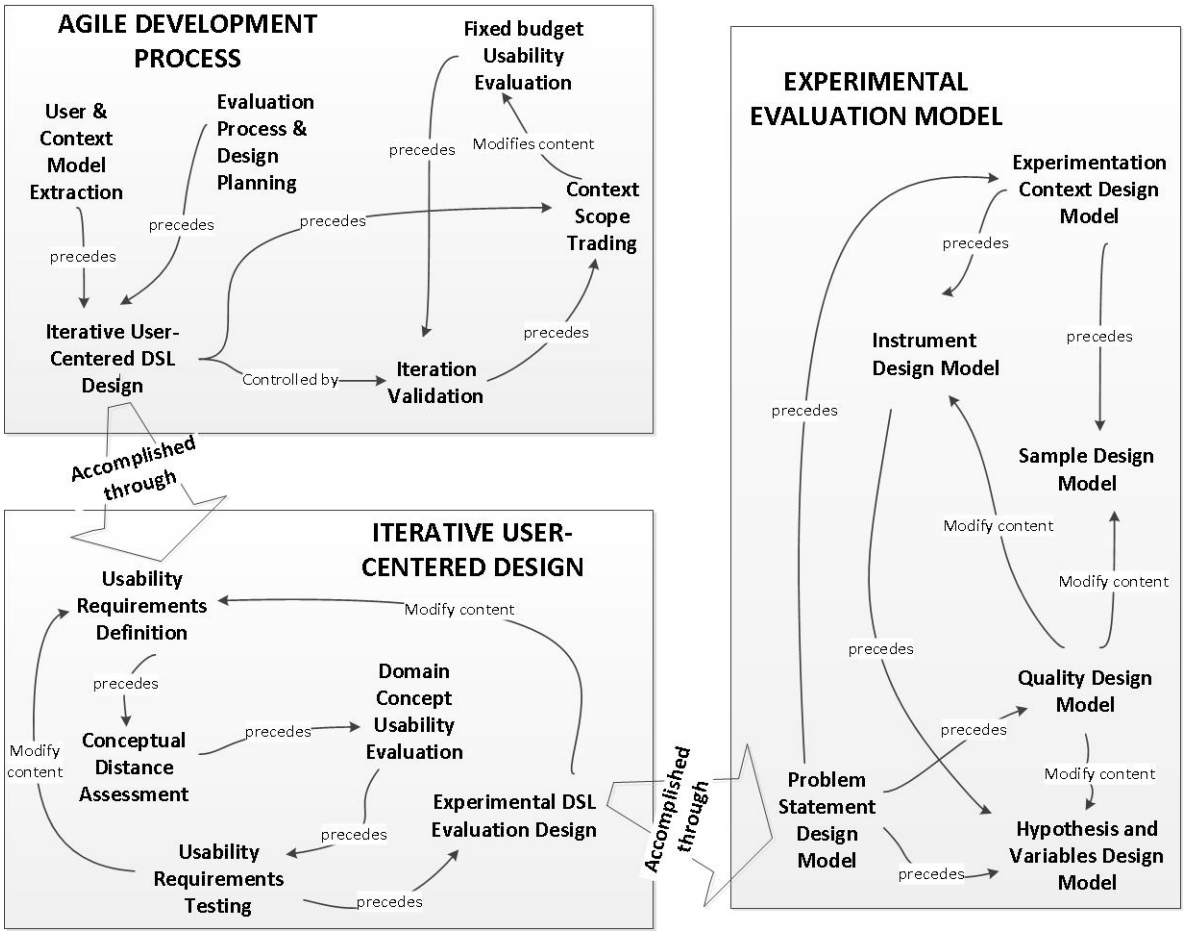


Figure 4.10: Pattern Language (taken from [21])

The goal of our systematic approach is to support DSL usability evaluations since the early stages of its development to prevent user-interaction mistakes, hence achieving a usable DSL by construction. The initial vision is detailed by a pattern language for evaluating the usability of DSLs that places the intended users of a language as the focal aspect of its design and conception by establishing formal correspondences for all stages between the DSL development process and the usability evaluation process. This iterative approach allows us to track the usability requirements and impact of recommendations, with a well-prepared evaluation process, allowing us to control the budget and the scope of the language's evaluation.

A pattern language is a set of inter-dependent patterns that provide a complete solution to a complex problem [47]. The main purpose of these patterns is to identify which commonly and successfully-used techniques for usability evaluation can be effectively applied in DSL design and guide the reader on the process of applying said techniques.

As such, apart from the provided examples, the main core of Known Uses and Examples we provide exist outside the realm of DSLs. This set of practices reflecting high-level guidelines are presented in Figure 4.10 and detailed in AppendixB. Our inter-dependent set of patterns is divided into the following three design spaces:

4.2.2.1 Agile development process

Agile Development Process breaks tasks into small increments and each iteration should fit in short time-boxes that typically do not last more than a month [142]. It promotes face-to face communication in workshops without impact of hierarchy roles of team members. All of them should take same level of responsibility that business and user needs are satisfied, by optimizing impact of evaluation feedback on language development. An appropriate iteration strategy that balances time invested into design of problem and its solution should be planned well with technical implementation. When goals are scoped and budget is fixed, we are ready to proceed to design and implementation activities that are guided by patterns given by Iterative UCD (Section 4.2.2.2).

This design space includes patterns devoted to project management and engineering of a DSL. Through organisation and planning of language development, this practice enables the control of evaluation activities, their scope (i.e. the context of use), the budget and tracking the success of the DSL.

These patterns are summarized as follow:

- *User And Context Model Extraction* (Section B.1.1). Before building a new DSL we should identify all intended user profiles and target context of use.

- *Evaluation Process And Design Planning* (Section B.1.2). Usability evaluations and experimental designs should be carefully planned through an experimental process model.

- *Iterative User-Centered DSL Design* (Section B.1.3). Introducing DSLs User-Centered methods allows us to achieve a productivity increase.

- *Iteration Validation* (Section B.1.4). By validating the iterations in time-box fixed intervals we can monitor progress and check if development is going in the desirable direction.

- *Context Scope Trading* (Section B.1.5). Short iterations require short and well scoped contexts.

- *Fixed Budget Usability Evaluation* (Section B.1.6). In order to reduce the cost of Usability validation and increase the validity of design decisions, the development team should plan development budgets according to the scope of iteration.

4.2.2.2 Iterative User-Centered design

It is necessary to engage the End Users in the language design in order to collect valuable information about their working scenarios and requirements [169]. In order to assess appropriateness of given concept design decisions it is necessary to identify meaningful quality attributes for each domain concept and its use. Metrics should be defined and calculated based on their dependency to designed concepts and should be in conformance with evaluation goals. Finally, they are expected to result with concrete hypotheses, tests, metrics, samples and statements that should be addressed and validated trough Experimental Evaluation Model.

Iterative UCD provides patterns for the engagement of the Expert Evaluator into the development process, with the intention of collecting relevant information concerning the Language Engineer perception of the problem solution and Domain Users' interpretation. Depending on the cognitive model instances to be evaluated, the DSL implementation technique or usability investigation technique, each design vary a lot.

These patterns are summarized as follow:

- *Usability Requirements Definition* (Section B.2.1). While building domain concepts, through direct interaction with Domain Experts, it is valuable to collect background information of the target users of each language concepts, in order to specify what usability means to them.

- *Conceptual Distance Assessment* (Section B.2.2). In order to understand how the design of the language's architecture impacts usability requirements, it is necessary to elect quality indicators and relate them to domain concepts.

- *Domain Concept Usability Evaluation* (Section B.2.3). Using metrics to analyze the metamodel's concepts representation allows the Language Engineer to reason on how different concept models impact the DSL's quality in use.

- *Usability Requirements Testing* (Section B.2.4). It is necessary to provide tests and evaluate if the current implemented features contribute to the defined goals.

- *Experimental DSL Evaluation Design* (Section B.2.5). When a release candidate version of the DSL for a specific target user group seems to be ready for deployment, an experimental usability validation should be performed with real users and real test case scenarios.

4.2.2.3 Experimental evaluation design

Experimental evaluation design supports the specification of experiment (e.g. hypothesis, tests, metrics, samples and statements) and is instantiated by evaluation model (Section 4.1) which is expected to support specifications for any usability investigation assessment. Example of model instances can be find in Appendix:

- *Problem Statement Design Model* (Section B.3.1).
- *Experimentation Context Design Model* (Section B.3.2).
- *Instrument Design Model* (Section B.3.3).
- *Sample Design Model* (Section B.3.4).
- *Quality Design Model* (Section B.3.5).
- *Hypothesis and Variables Design Model* (Section B.3.6).

4.2.2.4 Related patterns

There is a related line of work on HCI patterns, covering areas like ubiquitous systems [170], web design [195], safety-critical interactive systems [100], as well as more general interaction design languages [158, 175, 192, 199]. Although HCI has a large focus on Usability, the patterns available mainly avoid process patterns and prefer patterns that represent actual usable human interaction artifacts [138], like News Box, Shopping Cart or Breadcrumbs.

Spinellis [185] presents a pattern language for the design and implementation of DSLs. Contrary to ours, these patterns refer to concrete implementation strategies and not to the process of building the DSL or usability concerns. Gunter [92] presents a pattern language for Internal DSLs. These patterns mainly focus on how to map domain concepts to language artifacts and follow by implementing given artifacts with a GPL capable of supporting internal languages.

Much of our patterns are based upon Völter and Bettin's pattern language for MDD [203]. These patterns represent a well-rounded view of MDD but they do not explicitly account for the importance of Usability in DSLs and therefore do not give explicit instructions on how to test and validate usability of the end product. It is our opinion that our pattern language can be composed with Völter and Bettin's to produce a more complete version of a pattern language for MDD with usability concerns. To the best of our knowledge, ours is the only pattern language focusing on Domain Specific Language development process with user centered design.

As for usability, there are not many patterns or pattern languages available to cover usability concerns. Folmer and Bosch [72] developed a usability framework based on usability patterns to investigate the relationship between usability and software architecture. This work however has little relation to usability tests and to the development of usable software through usability validation. They instead map well know HCI patterns, such as Wizard, Multi-tasking and Model-View-Controller to quality attributes and usability properties. However, this is somewhat related to our Conceptual Distance Assessment pattern (Section B.2.2) and the framework could in theory be used to identify the mappings between domain concepts and quality attributes. Graham's pattern language for web usability [85] deals with usability evaluation and usability testing process.

However, these patterns are hard to follow due to the high number of patterns and lack of formal structure. Furthermore, Graham's patterns are targeted at web-based software. The pattern language most similar to ours is Gellner and Forbig's Usability Evaluation Pattern Language [79]. This pattern language is composed of thirty five patterns for usability testing. Of those, the Eight Phase pattern represents a set of eight stages of the process of usability evaluation. This is a similar approach to ours and has the merit of summarizing the process into a single pattern. However, the goal of the pattern is to disseminate usability evaluation for small scale projects while our pattern language considers small to large projects.

Chapter 5

The Usability Software Engineering Modelling Environment (USE-ME)

As already mentioned in Section 2.2, usability engineering aims to increase the awareness and acceptance of established usability methods among software practitioners. Knowledge of the basic usability methods is expected to enhance the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment. It is claimed to affect the user's performance and satisfaction. In Chapter 4 we introduced a systematic approach for usability evaluation of DSLs as a pattern language, and defined the experimental model for DSLs evaluation. In this Chapter we introduce usability engineering methods into the DSL life-cycle (presented in Section 2.1) by following an MDD SLE process.

The main concepts supporting the modelling approach are introduced as UML class diagrams. The flow of activities is described by UML activity diagrams. While discussing the proposed conceptual framework which is based on our systematic approach, we will use the following symbols:

- [] - main concepts i.e. classes in diagrams;
- «» - attributes and relations of the introduced concepts that are part of the diagram presented in a section,
- <*Italic*> - attributes and relations that are part of diagrams from other sections,
- {} - instantiation of the concept i.e. instance object;
- " - activity in a diagram
- *'Italic'* - decision in activity diagram

Similarly to other software qualities, usability evaluation should not be just added at the end of the development process. Following the discussion in Section 2, we argue that it has to be included in the development process from the beginning by accurately profiling the end-user and finding the right definition of the problem. The approach is to support usability evaluations, expressed as regular Expert Evaluator activities, in the SLE, backed by a conceptual framework that promotes the iterative UCD evaluation approach (proposed in Chapter 4).

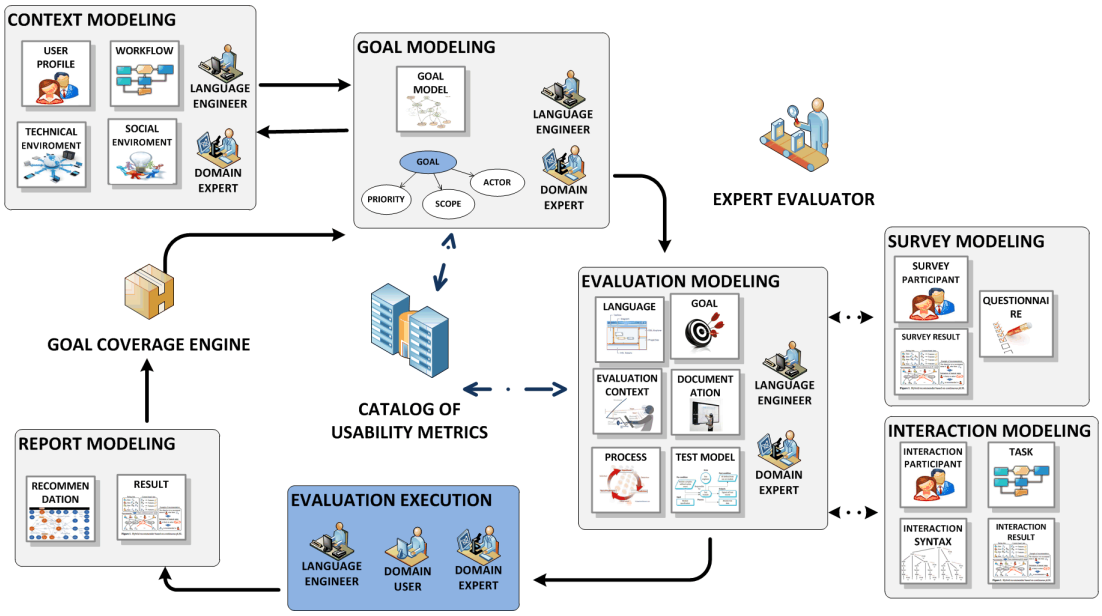


Figure 5.1: USE-ME life-cycle (taken from [26])

The Usability Software Engineering Modelling Environment (USE-ME) presents implementation of our systematic approach from previous Chapter and consists of the following modelling activities that are expected to be performed by the Expert Evaluator (see Figure 5.1):

- the **Context Modelling** to define the context of use for the DSL;
- the **Goal Modelling** that sets the objectives for the DSL;
- the **Evaluation Modelling** that instructs the usability experiments;
- the **Survey Modelling** that provides the support for survey/feedback collection;
- the **Interaction Modelling** that defines the interaction task under study;
- the **Report Modelling** that supports management of the collected data.

The goal specifications for the DSL under evaluation are underpinned by the **Catalogue of Usability Metrics** that describes usability goals and its possible measurements and treatments. This Catalogue helps to specify the relevant metrics and interpretation of

the results. Finally, the **Goal Coverage Engine** versions the iterative evaluations and defines context coverage for the validated goals. The Expert Evaluator in certain modelling activities is supported by artefacts and feedback provided by other DSL stakeholders.

In Figure 4.9 we introduced the usability evaluation process during the DSL development. The *domain engineering* phase of the process is supported by USE-ME Context and Goal Modelling activity. Further, during *design the language* phase Expert Evaluator is finalizing Goal Modelling and starting the Evaluation Modelling activity. Finally, *implementation and testing* phases are supported by Evaluation and Result Modelling activities.

We specified a supporting pattern language for our systematic approach in three design spaces in Section 4.2 (Figure 4.10). The USE-ME conceptual framework support the *Agile development process* within Context Modelling activity, which helps in prioritizing relevant context for the development cycle, therefore illustrating a scope and associated cost for intended evaluation process. The USE-ME Goal Modelling activity is essential in supporting the *Iterative User-Centered Design* space, as it supports the definition of usability goals, associated requirements, context dependent metrics and calculation of success, which is correlated with a scope for which it was validated. The Goal Coverage Engine supports the iterations by merging the obtained results of evaluation, produced by Report Modelling activity, and illustrates their impact on new goal model, by taking into account specified context. Finally, the Evaluation Modelling supports the *Experimental Evaluation Model* design space.

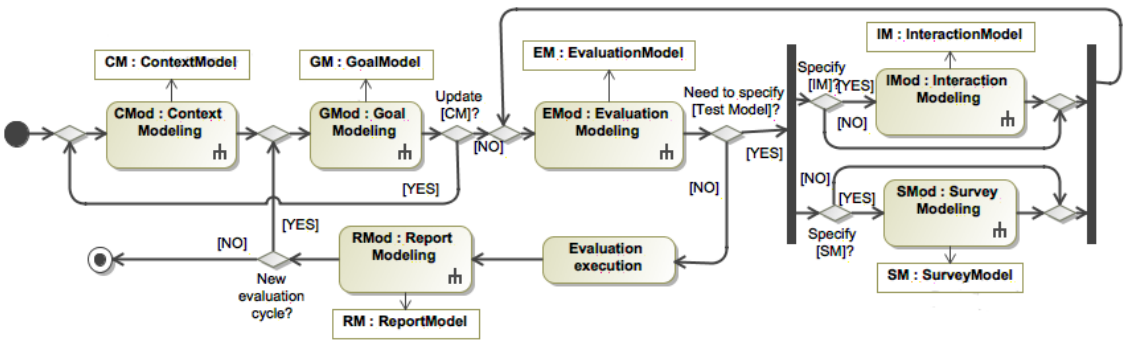


Figure 5.2: USE-ME activity diagram (taken from [26])

We present the process of usability evaluation using a USE-ME conceptual framework as an activity diagram (see Figure 5.2). First, it is necessary to produce a [Context Model (CM)], which supports the description of language context, based on which is possible to describe the usability goals as a [Goal Model (GM)]. The Language Engineer and the Domain Expert are involved in the 'Context Modelling' and 'Goal Modelling' activities, as they are expected to contribute to the interpretation of development artefacts (i.e. domain model, feature model, abstract syntax, etc.) and review of produced specifications. While specifying the goals and their scope, it is likely that new context elements which are relevant for the use of the DSL are found. In this case, it is necessary to *'update a CM'* and proceed with specifying a [GM] until we have at least one usability goal for which

only actor representing Expert Evaluator is responsible. Further, it is possible to define an [Evaluation Model (EM)], which highlights evaluation goals and their corresponding evaluation steps. The Language Engineer provides the developed artefacts (i.e. DSL, documentation, validation tests) and helps to prepare the evaluation. There is a *'need to specify a [Test Model]'* or reuse an existing one. The [Test Model] is crucial for the assessment process and can be defined as an [Interaction Model (IM)] or/and a [Survey Model (SM)]. These two modelling activities depend on the same [EM] and should be performed in parallel to complement each other. However, for certain types of evaluations, it is not necessary to develop both models. For instance, when performing a heuristic evaluation, a checklist implemented as an [SM] can be sufficient. When the [EM] is ready, we can proceed with the 'Evaluation execution' in which Domain Users are included as subjects, while the Language Engineers and Domain Experts help in the evaluation execution. The next step is to analyse the stored results of the test models and to create the [Report Model (RM)], that recommends a [GM] extensions and calculate a success factor of the evaluated usability goal. Finally, it is up to all DSL stakeholders to decide to continue to *'new evaluation cycle'* or finalise the assessment period. Ideally, this decision will eventually indicate the end of the development cycle.

5.1 Utility

In this section, we introduce the Utility package, which contains artefacts that are reused from the existing development process of DSL. It includes the following concepts (see Figure 5.3):

- **DSL** - represents a language artefact, which can have associated to it an *Abstract Syntax* and a *Concrete Syntax*. This can be seen as a reference object of the DSL under development and its progressive design implementations.

- **Existing GM** - goal model is a standard development artefact in MDD. If the DSL is evolving from the previous state, or usability analysis is performed in later phases of they development cycle like implementation or testing, we already may have an existing GM from which we should reuse the knowledge. We will not focus during this analysis on *Functional* and *non-functional* goals and requirements. They are seen as a part of an existing goal model. Instead, we are addressing usability goals and requirements which are often dependent on the satisfaction of existing goals and requirements.

- **Profile Template** - is a classification artefact for a user profile. It is used for categorization of user profiles and specifying their features as a *Logical Expression* (e.g. regex, ocl expression, Boolean). These expressions contain a list of concrete (e.g. 'age'>7) or abstract (e.g. 'age'=int) specifications. Expected background experience for profile can be specified by required skills, which sometimes are predefined for certain working/study position.

- **CEVariable** - is a variable describing the environment of the language under development. Their elements are parts of the architectural design (e.g. feature diagrams), or

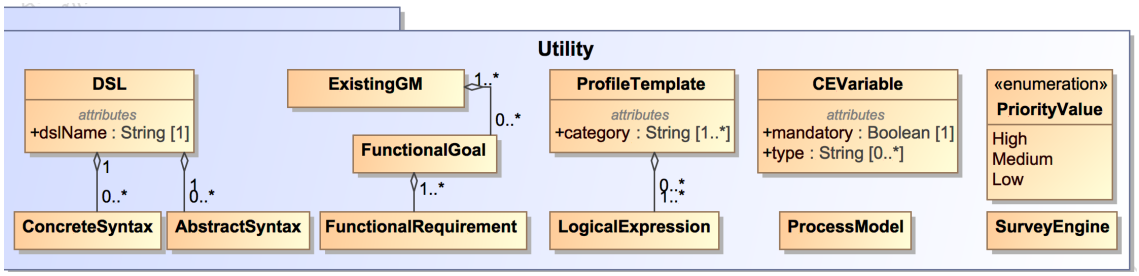


Figure 5.3: Utility package (taken from [26])

the specifications of language supporting equipment and dependent tools (e.g. sensors, operating systems, interaction equipment).

- **Process Model** - is an artefact which refers to business process models designed during DSL development. Additionally, specifies the experiment processes designed as a part of USE-ME conceptual framework.

- **Survey Engine** - represents an artefact which is connected to existing survey platform (e.g. Google Forms, Survey Monkey, etc.).

- **Priority Value** - represents predefined priority values, which are set to be 3-scaled with values [High], [Medium] and [Low].

5.2 Context modelling

As pointed in the previous chapter, the evaluations are context dependent. In consequence, the DSL's intended Context of Use should be specified when answering the following questions:

1. Who will use the DSL?

2. Where will the DSL be used?

3. How is the DSL expected to be used?

We argue that it is sufficient to describe the **Context Model (CM)** with the following concepts (see Figure 5.4):

- **User Profile** - helps us to define **who** will use the DSL. The user modelling activity [119] customises and adapts the language to the user's specific needs. Each profile can be instantiated as a subcategory of the main stakeholders during the DSL evaluation, namely {Language Engineer}, {Domain Expert}, {Domain User} and {Expert Evaluator}. Also, it stores the «priority» regarding the current evaluation needs (in the ongoing development cycle) expressed as an enumerated <*Priority Value*>. Any additional [User Profile] is justified by a «classifier», presenting any <*Logical Expression*>, which justifies new «sub-Profiles». Usually, during this classification, we categorise new profiles by particular «parent» characteristics into at least two distinct sets.

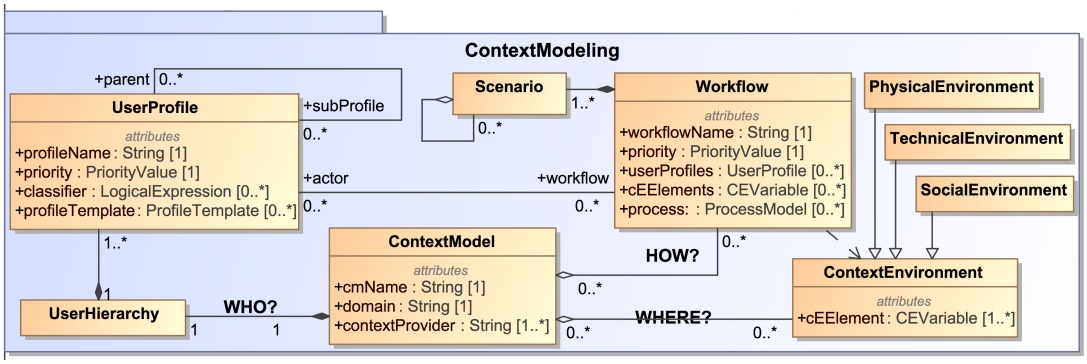


Figure 5.4: Context Model class diagram (taken from [26])

- **User Hierarchy** - is the prioritised categorization of the [User Profile] presented by the diagram in which each «subProfile» inherits the properties of its «parent». These properties are documented using <*Profile Templates*>, and they describe characteristics such as the expected background information (e.g. demographic data, education, special needs/disabilities) and relevant experience with computing and domain activities (e.g. expected knowledge sets, ontology). The [User Hierarchy] represents the user-centered meta-knowledge which categorises profiles and identifies shared semantics.

- **Context Environment** - describes **where** the DSL is to be used. Namely, it can be defined as:

- *Technical Environment* - specifies the information about the user's system usage. In other words, software, hardware and network environment that describe the users': i) working equipment (e.g. interacting device specification); ii) DSL operating equipment (e.g. storage, calculation libraries); iii) operation systems; iv) supported platforms; v) frameworks; vi) dependent software tools; vii) and, technical usage conditions (such as display capabilities, connection bandwidth, etc.).

- *Social Environment* - expresses the relation of the DSL to the users' working environment concerning the social context i.e. situational environment [94]. The referred situations are used to model a conceptual framework for representing a given the social context of the semiotic environment in which the users exchange meaning. This model recognises the fact that technology is not developed in isolation but as a part of the wider organisational environment.

- *Physical Environment* - describes the working equipment and organisation of the physical environment in which the user interacts with the system, as well as the models of the physical systems and their natural environment that are being affected by the use of the DSL.

The specification of the technical, social or physical elements is stored as a «cEElement» which is represented as an environmental variable <*CEVariable*>. The definition of the [Context Environment] instances can be supported by the integration of existing requirement engineering approaches that support traceability of the environmental variables during the software development [133].

- **Workflow** - describes **how** the DSL is to be used by documenting prioritised user workflows, which reflect a group of tasks relevant for specific «userProfiles» or environment elements i.e, «cElements». The [Workflow] can be defined by using the existing tools for modelling processes (e.g. BPMN [4], UML diagrams [73], or by the user requirements notation ITU standard [8]). These models are referred by the «process» attribute, which reflects the *<ProcessModel>*

- **Scenario** - represents a concrete task, i.e. use case, that produces the possible paths and outcomes. Each [Scenario] describes the user interaction activities, executed by the specific [User Profile] or «cEElement» from [Context Environment]. The [Scenario] can be further decomposed in more basic functional actions inside of the system that can be connected to a DSLs domain model, preferably to its concrete or abstract syntax features.

The Context Modelling activity, specified by the diagram in Figure 5.5, engages all stakeholders included in the development. It starts with 'User Hierarchy prioritising', for which initial [User Hierarchy] is represented by DSL stakeholders, if not specified differently. This prioritisation helps stakeholders to decide the usability evaluation is necessary, and also to indicate from which stakeholder's perspective. Usually, the high priority for the End User profile justifies the investment in a reactive experimental approach during the evaluation. Furthermore, it is necessary to perform 'User Profile classification' during which the relevant «classifiers» are identified, resulting a new [User Profiles] in the hierarchy. The parallel activity to this user identification is a 'Context Environment definition', during which the physical, social and technical environmental elements, i.e. «cEElements», are specified. Finally, after knowing who will use the DSL and where it is going to be used, the expert evaluator can describe why the DSL will be utilised by [Workflows] created during the 'Workflow definition'. It is likely that the evaluation expert will identify at this point missing context element specifications, or the need for a new [User Profile] classification. In this case, when the process reaches the decision point *'Need to extend CM?'* it is possible to return and extend associated models. Finally, when there are no new insights after the Workflow specification, the Expert Evaluator is ready to finalise this activity.

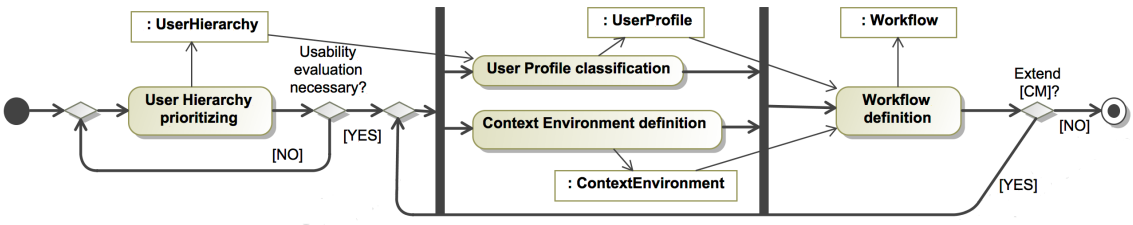


Figure 5.5: Context Modelling activity diagram (taken from [26])

5.3 Goal modelling

The **Goal Model (GM)** specifies objectives that a user may have while using the DSL. This model describes '**why** is the new language being developed?', and this makes it a central part of the USE-ME conceptual framework through which we are expected to model context dependent goals and trace the success of the DSL under development (see Figure 5.6).

Goals capture, at different levels of abstraction, the various objectives the system under consideration should achieve [196]. A goal model is a well-known element of requirements engineering that is also widely used in business analysis. They have been used in SLE to model non-functional requirements [58] and early requirements for software systems. There are already several approaches for goal modelling; namely, KAOS [197], i* [214] and Tropos method [82]. These approaches already support the core concepts of goal modelling, i.e., goal hierarchy and quantitative and qualitative relations. However, they should be specialised and, in some way extended to serve our research objectives. Usability is found in these approaches as a non-functional requirement or a soft goal. In our approach usability is meant to be highest level objective (i.e. goal) for the developed DSL.

The [GM] of USE-ME can represent extensions to <*Existing GM*> of the language, or can be built from scratch. It is described by **Usability Goal** using a GOMS (Goal, Operators, Methods, Selection) [66] and/or GQM (Goal, Question, Metric) analysis [198]. The root [Usability Goal] of the [GM] is the Quality in Use, and represents «parentGoal» of any newly defined «subGoal». Goals are characterised by:

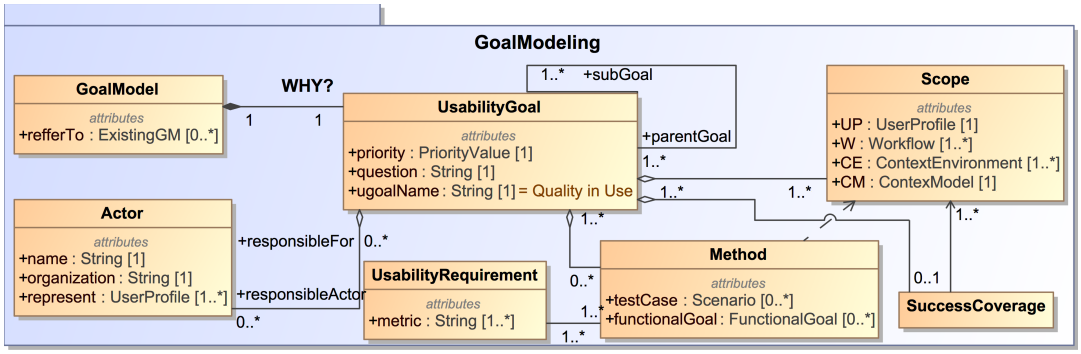


Figure 5.6: Goal Model class diagram (taken from [26])

- **Actor** - is a specialisation of DSL stakeholder, namely an instantiation of a [User Profile] from the [CM], to which are assigned the responsibilities («responsible For») for the associated [Usability Goal]. The resulting responsibility model clearly distinguishes stakeholders to which a [Usability Goal] is related (i.e. [Scope]) from the ones that are responsible for achieving it (i.e. [Actor], «responsibleActor»).

- **Scope** - that is described by the instance of [CM] for which [Usability Goal] applies. By default, each goal of [GM] applies for the complete [CM], if it is not specified differently

(e.g. specific <*User Profile*>, *Workflow*> or *Context Environment*>). The priority level of [Usability Goal] can be inherited from the [Scope] to which it is related to.

- **Method** - defines the measurable requirements (i.e. *Usability Requirement*s) that contribute to the achievement of the goal. It consists of «testCase»s taken as individual <*Scenario*>s from [Scope]). These «testCase»s can be used to evaluate the requirement. A [Method] is often dependent on the development stage of a DSL and the context of the planned evaluation. For instance, during early evaluations without a functional prototype, we can evaluate the readability and understandability of the design; when assessing early ideas, the focus can be to evaluate the feasibility of the language construction or usage process, and finally, when functional prototypes become available, we can perform experiments with using a DSL. A [Method] is associated with «functionalGoal» that should be tested by Language Engineer. The <*Functional Goal*> represents the functionalities that are to be provided to support the execution of a «testCase».

Success Coverage - reflects the evaluated context coverage and can be represented by the percentage of the [CM] (i.e. [Scope]) to which each success factor applies. As we are evaluating usability, we find that the metric representing the score for the evaluated scope can also rate on the scale from -1 to 1, where the (1) indicates that measured experience was positive, while (-1) indicates a negative experience while using the given DSL. Further, the representation of success can use Kiviat diagrams [19] reflecting different requirements that were taken into consideration, or by a single project dependent indicator predefined by stakeholder [194].

The Goal Modelling activity, specified by the diagram in Figure 5.7, starts with identifying if there is an *'Existing [GM]?'*. In case there is a [GM], previously created within USE-ME or another context, the Expert Evaluator performs '[GM refactoring]' regarding the evaluation Priority and a structure and identifies if there is a need to 'Change or introduce a new [Usability Goal]'. Further, the [Usability Goal] is changed/introduced during 'Goal specification' activity during which Expert Evaluator consults Usability Catalogue (Section 5.8). In parallel, a [Scope] is associated with the goals during 'Context selection' and an [Actor] during 'Responsible actor selection'. Next, it is necessary to verify if the *'Single [Actor] is responsible?'*, meaning that the goal is decomposed into a «subGoals» for which only one [Actor] is «responsibleFor», and by this making it ready for evaluation. This is followed by 'Functional Goal association' during which all the functional prerequisites that need to be verified for evaluation are provided by the Language Engineer and associated with [Method] justifying [Usability Goal] evaluation. In parallel, the Expert Evaluator defines an evaluation [Method] during 'Measurable method specification', which aggregates [Usability Requirements] and «testCases». This activity requires consultation of the Usability Catalogue, to reuse the existing metrics if possible. Finally, if the *'[Usability Goal] is evaluated'*, then it is feasible to perform a 'Success Coverage calculation' that indicates the scope for which the goal was evaluated, as well as the evaluation results.

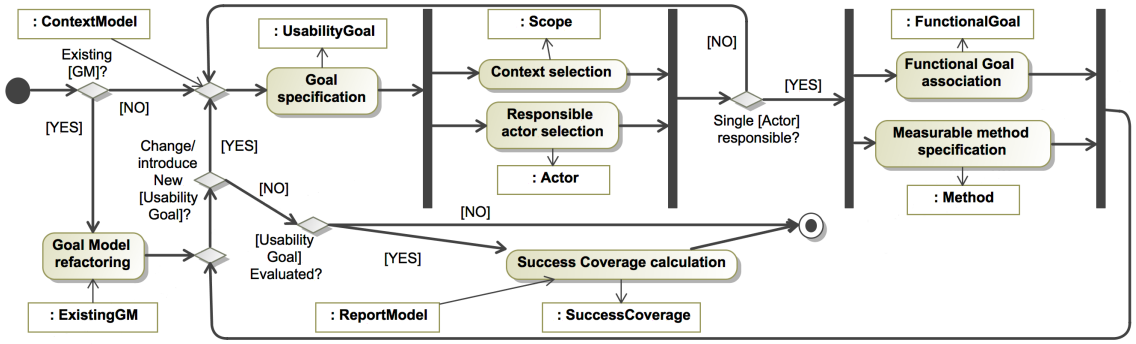


Figure 5.7: Goal Modelling activity diagram (taken from [26])

5.4 Evaluation modelling

Although DSLs' development process is not the same as the one of the UIs, typically DSLs have an explicit underlying model (thanks to meta-models or grammars), while UIs models are usually implicit in their implementation. In general, there is no distinction between DSLs and UIs from the end users point of view, so their evaluation should essentially validate HCI [18].

Therefore, we can reuse the common techniques used for UI usability evaluation; e.g. UCA (Usability Context analysis) [37], MUSiC (Metrics for Usability Standards in Computing) [40] and MAGICA [6] methods and tools. UCA ensures that the user-based evaluation produces valid results, by specifying how important factors are to be handled in the assessment. MUSiC and MAGICA provide modules that can be used for measuring user satisfaction, user performance, cognitive workload, task completion time and analytic measurement.

The evaluation modelling activity is expected to support the application of techniques mentioned above that are represented by the **Evaluation Model [EM]** (see Figure 5.8). [EM] expresses the purpose of evaluating a certain objective (an instance of [GM]) for a DSL in the particular context (an instance of [CM]). Therefore, the prerequisites for the evaluation modelling are to have a [GM] and [CM] for a DSL under evaluation. This activity is supported by modelling of:

- **Evaluation Goal** - defines the experimental «hypothesis» and «researchQuestions». It is related to «goal»s specified in [GM] and inherits its <*Methods*> which can be introduced as «measurements».

- **Language** of experimental study - is the <*DSL*> under evaluation. The experimental modelling supports more than one [Language] to enable comparative evaluations (e.g. with the baseline that can be an alternative system or previous DSL version).

- **Evaluation Context** - specialisation of the [CM] that describes the <*User Profiles*>, <*Workflows*>, and <*Context Environments*> taken into consideration during execution of the experiment. This model preferably should reflect the intersection of the [CMs] that specify [Languages] of the evaluation study, in which each [CM] reflects the <*Scope*> of

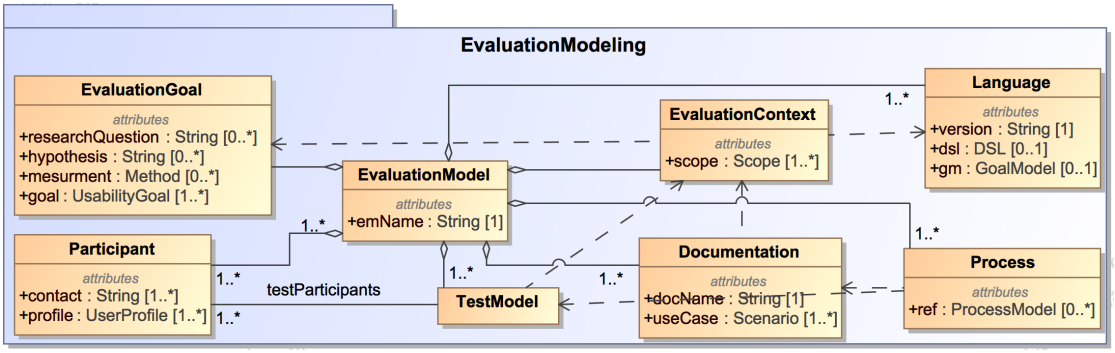


Figure 5.8: Evaluation Model class diagram (taken from [26])

[Evaluation Goals].

- **Participant** - refers to the actual participants of the study, which are expected to match the [User Profile] included in the [Evaluation Context]. It also stores contact information of experiment subject.

- **Documentation** - presents teaching materials for the [Language] under study, e.g. videos, guided examples with image annotations, presentations. These materials can be stored in a shared document repository with version support (such as a Wiki).

- **Test Model** - describes the usability testing activities that are not learning treatments, i.e. questionnaires, interviews, and observations. The creation and execution of these models are supported by the USE-ME sub-activities of survey modelling and interaction modelling, described in the following sections.

-**Process** - defines the concrete design for evaluation by modelling the activities that should be performed with specific [User Profile] that is described by the [Evaluation Context]. The choice of appropriate treatment is guided by its probability to evaluate the [Evaluation Goal]. These activities are represented by [Documentation] and [Test Models]. They specify the flow of learning treatment activities that are modelled in the evaluation [Process].

Evaluation modelling activity instantiate experimental model of our systematic approach, defined in Section 4.1.2. *Problem Statement Design* (Figure 4.2) and *Hypothesis and Variables Design* (Figure 4.6) are encapsulated in the [Evaluation Goal]. *Context Design* (Figure 4.3) definition is supported by [Evaluation Context] and [Process]. *Instrument Design* is supported by [Test Model], or namely [Survey Model] or/and [Interaction Model]. Finally, *Sample Design* (Figure 4.5) of experimental model is supported by [Participant] and [Language] concepts.

When preparing an evaluation (see Figure 5.9), the evaluation expert performs an 'Evaluation language selection' where the DSL under development is chosen to be an evaluation object. Next, he defines the experimental objective and participants during 'Evaluation Goal definition' and 'Participant selection' activities. He also decides if he will perform a 'Comparative evaluation?' and, if so, which other language he will use ('Baseline selection'). The 'Evaluation Context specification' activity defines the context

that is considered during evaluation, taking into consideration «profile» of selected [Participants] and the «scope» of the [Evaluation Goal]. This activity can be extended by the context modelling activity if the existing [CM] is missing relevant information. Before the 'Evaluation execution', it is necessary to produce [Documentation] during the 'Teaching Material creation', define evaluation [Process] by 'Process specification' and prepare the [Test Model] during 'Test Model creation'. The 'Test Model creation' activity selects a previously created [Test Model] or calls the Interaction Modelling or Survey Modelling to produce the new or refined [Test Models].

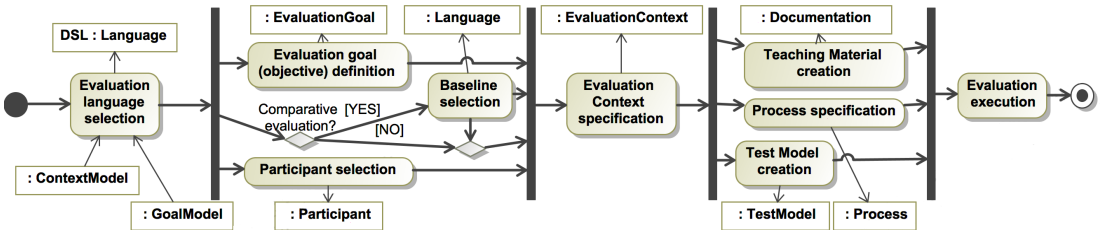


Figure 5.9: Evaluation Modelling activity diagram (taken from [26])

5.5 Interaction modelling

The [Test models] that are defined as **Interaction model [IM]** (see Figure 5.10) encapsulate summative methods for measuring usability over concrete tasks that involve interaction with at least one [Language] (i.e. DSL under evaluation or its alternative). The [IM] supports capturing of the predefined events and providing statistics about their occurrences. It is described by:

- **Interaction Syntax** - reflects the interaction elements from the version of the [Language] of experiment study (for instance DSLs abstract and/or concrete syntax model, or feature model).

- **Task** - «useCase» taken from the «scope» of [Evaluation Context]. It is documented in [Documentation] and represents a concrete task for which the interaction will be analysed.

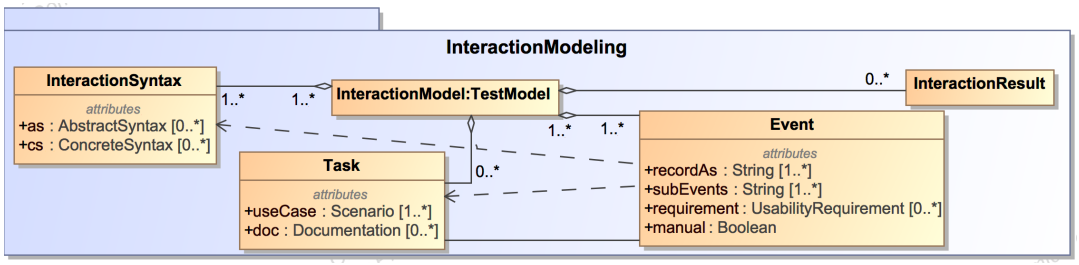


Figure 5.10: Interaction Model class diagram (taken from [26])

- **Event** - accounts for the type of data («subEvents») that will be captured from different interaction devices. It describes how it will be recorded and can refer to single events

(e.g. eye movement, mouse click, biosignal, gesture), or to the particular sequence of interaction and/or the sequence of reactions (themselves or as a consequence of interaction sequence). Also, events can be associated with the «requirement».

- **Interaction Result** - includes the statistical analysis and results of the executed interaction model, where participants use the [Language] of experiment study to perform a [Task].

Interaction Modelling (see Figure 5.11) starts with 'Interaction Task definition', that is interdependent on 'Interaction Syntax analysis', as the task is to be solved by the use of the analysed syntax, following the [Workflow] that is included in the [Evaluation Context]. Based on the [Task], the evaluation expert is ready to perform 'Events specification' during which [Event]s are defined, as well as the way to capture them. Further, he performs the 'Interaction Participant assessment' activity where the experiment [Participants] are assigned to the [IM]. In parallel, the scheme for [Interaction Result] is prepared by 'Interaction Result formatting' activity. Finally, when the [SM] is complete, it is sent to 'Interaction execution' during which results are automatically stored in the [Interaction Result] model.

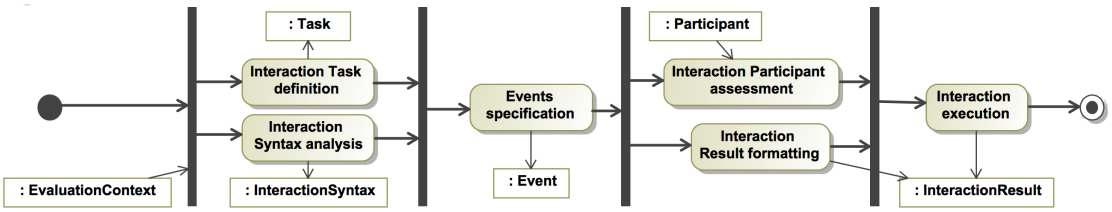


Figure 5.11: Interaction Modelling activity diagram (taken from [26])

5.6 Survey modelling

The survey methodology is a field of applied statistics that studies the sampling of individual units from a population and the associated survey data collection techniques, such as questionnaire construction and methods for improving the number and accuracy of the response to the inquiry. A survey is not just the instrument (the questionnaire or checklist) for gathering information. It is a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes, and behaviour [117].

In the case of USE-ME, the Survey Modelling activity supports formative methods for measuring usability. The **Survey Model [SM]** (see Figure 5.12) can correlate to any existing [Survey Engine] that automates response collection (e.g. Google Forms, Survey Monkey, mySurveyLab, etc.). It is described by the following concepts:

- **Questionnaire** - defines a particular set of **Question**s (inquiries) that can be provided in different forms; on-line, by phone or personal interview, pen and paper, and in advanced interaction environment (a testing environment that supports additional interaction equipment that can capture eye movements, gesture, biosignals). Generally,

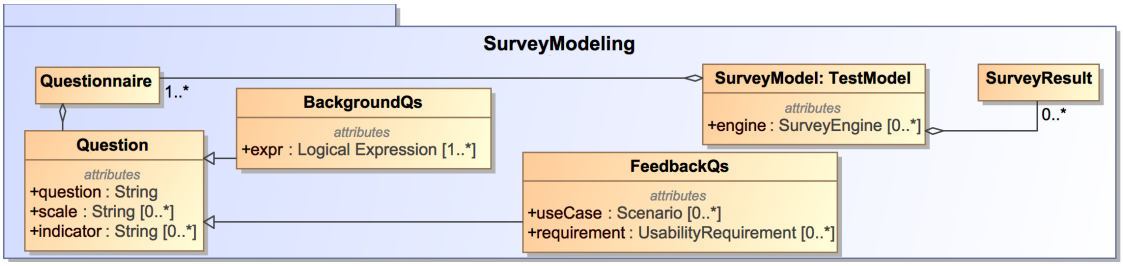


Figure 5.12: Survey Model class diagram (taken from [26])

[Question] has an associated concrete «scale», represented as a list of String values. It can also have a defined «indicator» that helps to get fine-grained variables that can help during the results analysis. The [Question] can be defined as:

- **Background Qs** - designed to collect the information about the participant (e.g. demographic data, education, special needs/disabilities). Each question is related to at least one <*Logical Expression* of participant's <*User Profile*>.

- **Feedback Qs** - collects the opinions and reactions about what is being tested i.e. the DSL, its baseline or alternative. In our case it would be expected to support existing appropriate methods such as inspections (usability heuristics [154], quality requirements for DSLs [121], Framework for Qualitative assessment of DSL [110]) and notation assessments (Cognitive dimensions of notations framework [41], Physics of Notations [151]). Each question, or a checkpoint, can refer to concrete «useCase» and/or «requirement».

- **Survey Result** - includes the statistical analysis and results of a survey, that can be generated automatically if <*Survey Engine*> was used, and further customised (e.g. merged results from different questionnaires, formatted, statistically analysed, etc.).

The creation of [SM] (see Figure 5.13) starts with asking if *'New questions?'* are needed to be defined for the survey. In the case they are needed, the evaluation expert decides if the questions to be defined are '[Background]', leading to 'Background question definition', or '[Feedback]', leading to 'Feedback question definition'. When there are no *'New questions?'*, the expert evaluator is performing 'Survey participant assessment' where the experiment [Participants] are assigned to the [SM]. In the same time, the scheme for [Survey Result] is prepared in the 'Survey result formatting' activity. Finally, when the [SM] is complete, it is sent to 'Survey execution' during which results are automatically stored in the [Survey Result] model.

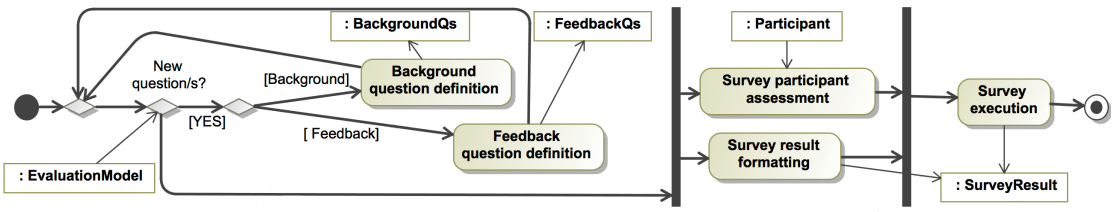


Figure 5.13: Survey Modelling activity diagram (taken from [26])

5.7 Report modelling

The report modelling activity helps on the construction of final reports for experimental assessment specified by the [EM] and encapsulate the results and improvement suggestions into the **Report Model (RM)** (see Figure 5.14). It consists of:

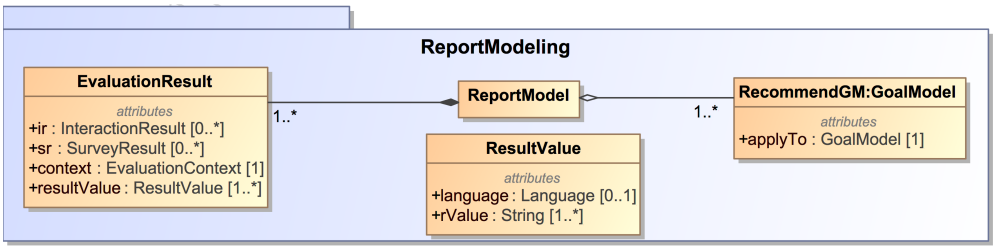


Figure 5.14: Report Model class diagram (taken from [26])

Evaluation Result - is created based on analysis of the result models from different <*Test Models*> (i.e. <*Survey Result*> and <*Interaction Result*>). It represents the interpretation of results over a predefined «context» that is related to the evaluated goal.

RecommendGM - is a recommended [GM], which include updates (changes over or new goals or context elements) to previous [GM] of the evaluated DSL, referred with «applyTo».

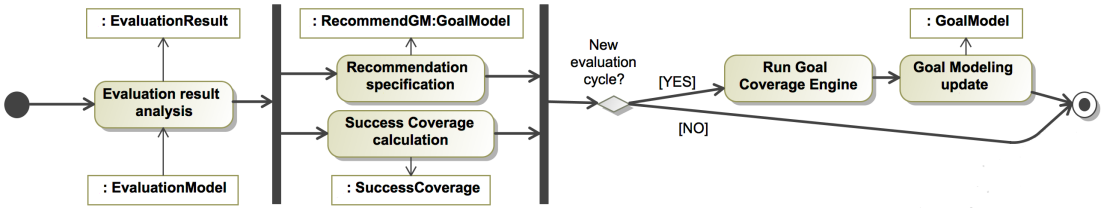


Figure 5.15: Report modelling activity diagram (taken from [26])

In Figure 5.15 we describe the report modelling activities. The first activity is to perform the 'Evaluation result analysis'. This includes reasoning about the correlations among different factors assessed by experimental instruments (namely, [SM] and [IM]). Based on these results, the expert evaluator performs a 'Recommendation specification', by designing a [RecommendGM]. At the same time, the evaluator can calculate a [Success Coverage] for the recommended [GM], which helps in making decisions how to redesign goals. Finally, when the recommendation and results are ready, all stakeholders should decide if they should enter *'New evaluation cycle'*. When starting a following cycle, the expert evaluator should 'Run Goal Coverage Engine' which validate that all rules are preserved and support a merge of recommended [GM] with the initial [GM]. Finally, during 'Goal modelling update' recommendations are accepted or rejected.

5.8 Catalogue of usability metrics

This catalogue is seen as a structured knowledge base about the DSLs usability evaluations. It has the purpose to help during two crucial activities of the USE-ME conceptual framework:

- **Goal modelling**, where it is used to find existing specifications of the usability goals and requirements. These specifications can be based on standards [104], or existing frameworks which address the evaluations in general [88, 110, 151]). On the other hand, catalogue is intended to store examples of methods and metrics which were used to test usability requirements for existing DSL evaluations.
- **Evaluation modelling**, where it is expected to support the choice of treatments for evaluation process activities based on a diversity of goals and the number of available participants, technology, etc. Further, it is meant to register the instantiation of quality in use metrics [101] for selected evaluation objectives and enhance a metric reuse and improvement.

To obtain the first structure of this knowledge base will be necessary to summarise the state-of-the-art research trends, as well as to categorize the proposed approaches, techniques, tools and methods for domain analysis and evaluation phases. Further, it is necessary to obtain USE-ME model instantiations for several evaluation assessments. The structure of different instantiations is expected to fit into *Quality Design* model (see Figure B.22) from the experimental model of our systematic approach.

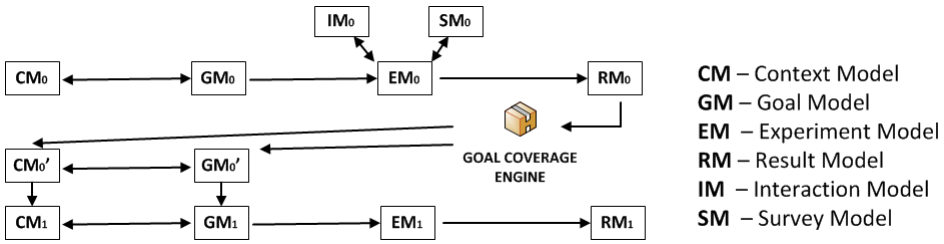


Figure 5.16: Model to model transformations supported by Goal Coverage Engine (taken from [26])

5.9 Goal coverage engine

This engine should provide version support of the USE-ME project models and support their updates by using model merge approaches (see Section 9.5, Annex V), while checking the predefined rules. The Goal Coverage Engine supports all DSL stakeholders to trace different states of models for different versions of the DSL or their assessments, as well as a knowledge base. For each new iteration cycle of the DSL development, the stakeholders need to confirm that the context and goals are still the same, or update

them if they have changed. This engine should also check the validity of specific rules for removing/merging/updating USE-ME models. Finally, this engine can be seen as a transformation engine that performs the updates of GMs based on the recommendations of the Report Model and finds the scope coverage for which the goals are validated based on a context instance (see Figure 5.16).

Chapter 6

Feasibility study with the USE-ME prototype

In a Chapter 5, we described the implementation of our systematic approach in the form of a conceptual framework, for which we present a feasibility study in this Chapter. First, we present the implementation architecture of our prototype. Further, we illustrate an instantiation of the prototype models on an industrial case study. Finally, we perform a pilot empirical evaluation of the implemented tool on four DSL development projects. This illustrates the feasibility of capturing all the relevant information for conducting a DSL evaluation, and how it can be used in results analysis.

6.1 Implementation architecture

The USE-ME prototype [27] was developed as a specialization of our conceptual framework (see Figure 6.1). The USE-ME conceptual framework was formally specified on Cameo Systems Modeler[1], a cross-platform collaborative model-based systems engineering environment. This platform enabled us to produce UML [2] compliant models and diagrams. Namely, we specified the main concepts with class diagrams and the process as activity diagrams. These diagrams served as a general specification of the abstract syntax for the USE-ME conceptual framework. They were reviewed by the Automated Software Engineering research group of NOVA-LINCS during a presentation session and trough individual questionnaires.

The USE-ME prototype[3] was developed using the Eclipse Modeling Framework (EMF)[4],

[1] https://www.nomagic.com/products/cameo-systems-modeler (accessed September 19, 2017)
[2] http://www.uml.org/ (accessed September 19, 2017)
[3] https://github.com/akki55/useme (accessed September 19, 2017)
[4] http://www.eclipse.org/modeling/emf/ (accessed September 19, 2017)

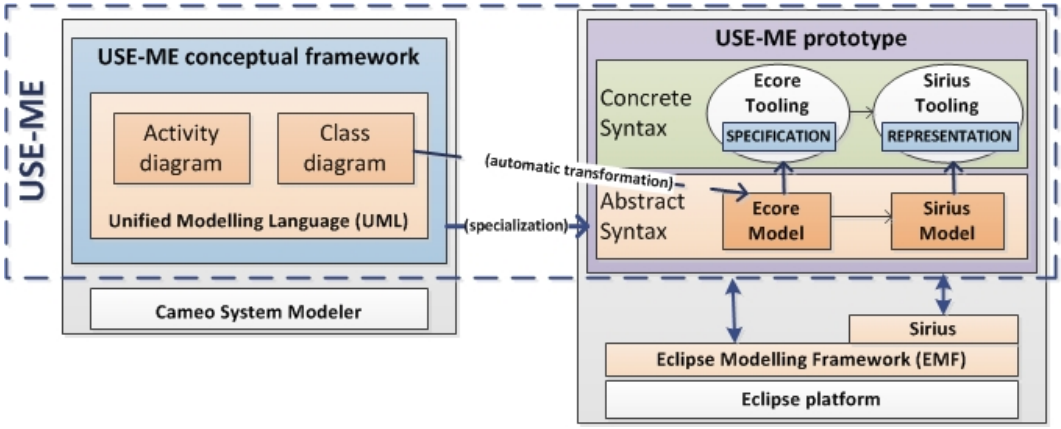


Figure 6.1: USE-ME architecture (taken from [26])

which is an Eclipse-based modelling framework for building tools and other applications based on a structured data model. From an XMI model specification, EMF provides tools and runtime support to produce a set of Java classes for the model. The EMF framework includes a meta model (Ecore) for describing models and runtime support for the models. The class diagram specified in Cameo System Modeler was transformed to an Ecore model. Due to constraints in the target meta-modelling tool, we had to adapt the model given in pure UML in order not to lose information from the original contents and to address restrictions required by Ecore (e.g. necessity of containment relationships).

Further, we used Sirius [5], a platform for developing and using graphical model editors, which is also based on the Eclipse Platform, and in particular the modelling stack based on EMF. The Sirius platform is domain-agnostic, in that it can be used by modellers for any business domain as long as they can describe it using EMF. In our prototype, we used Sirius to declare the visual representation of models instantiated in Ecore. The USE-ME architecture as described in this document supports modellers to design the USE-ME instances in an EMF generated editor with Ecore tooling. It is also possible to redefine and preview the implemented representations by Sirius.

6.2 Case study

We dedicate this Section to illustrate the USE-ME conceptual framework with a case study about a free web-based programming language, named *Visualino* (see Section 9.4, Annex IV), for rover-like robots. The work was developed under the collaboration between the group ASE NOVA/LINCS and Artica[6], a company that specialises in the development of robotic and audio-visual solutions. Visualino was developed to lower the barriers between this hardware technology and children. The goal is for children to program the robot using Visualino. The Visualino specification is then automatically translated into the

[5] https://eclipse.org/sirius/ (accessed September 19, 2017)
[6] http://artica.cc/ (accessed September 19, 2017)

Arduino textual code. This approach avoids the inherent complexity of programming directly the Arduino which requires a steep learning curve. The following description refers to the design of an empirical study conducted during the second development iteration of Visualino[7], where the USE-ME conceptual framework was applied systematically.

During the description we use following symbols;

- [] - instantiation of the concept i.e. instance object;
- «» - property of the instance object;
- {} - property value of the instance object;

6.2.1 Context model instantiation

The Context modelling starts with prioritising the initial user hierarchy, which is represented by the DSL's stakeholders. Therefore, we are having a [DSL Stakeholder] as a 'root' element of the hierarchy, and its subProfiles defined as [Domain Expert], [End User], [Expert Evaluator] and [Language Engineer] (see Figure 6.2).

For the case of Visualino, the target users are children, which usually are not experienced with programming. As such, the usability evaluation of the language with its end users is highly important. We set up the priority of its [End User] to be {High}. Having, a lower priority value associated with other stakeholders indicates that the investment in the usability evaluation of these profiles is not as important during the current development cycle.

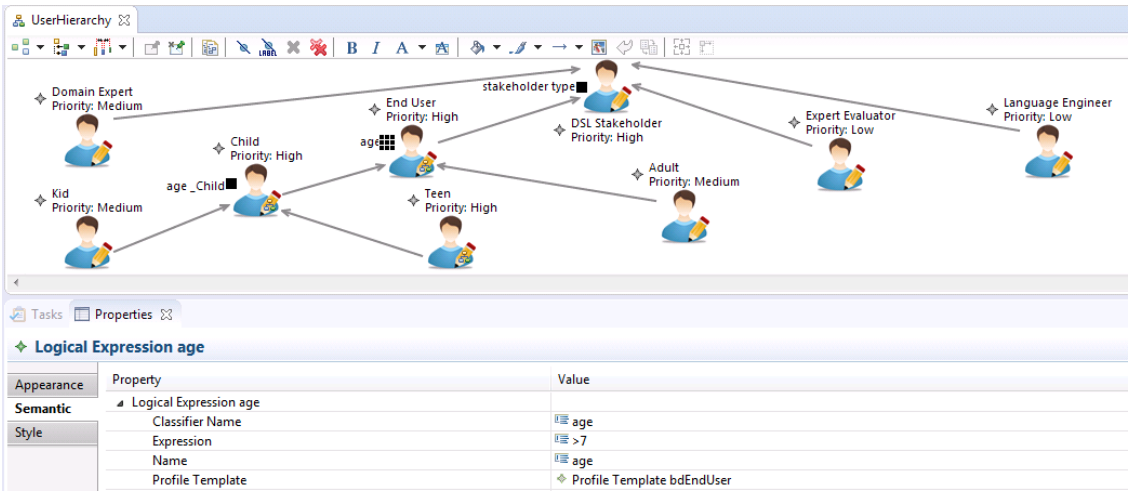


Figure 6.2: User Hierarchy (taken from [26])

However, as the language develops further, the investment into the evaluation with other user profiles may become more important. For instance, the development iteration may focus on the language extension that supports the [Domain Expert] to specify the environment configurations for different robots. Then, the priority for the [Domain Expert]

[7] https://sites.google.com/view/vl-empiricalstudy/home (accessed September 19, 2017)

profile will be updated accordingly. In this case, the priority of the [End User] should stay the same. Setting up a 'lower' priority is reasonable only if the initial purpose of the language has changed. For example, this may occur if the language 'fails' to be adopted by the initial audience, and the opportunity for further development presented itself in a different context.

After the initial prioritisation, we classify user profiles and define profile templates (see Figure 6.3). Profile templates are categorised as a {Background Demographics} and {Background Experience}. They are captured with logical expressions. These expressions reflect profile characteristics and their values, concrete or as measurable scales. For instance, the expression «language» indicates that [End User] is expected to speak {Portuguese} or {English}. On the other hand, «programming» is a characteristic that can be measured by {ordinalscale(experience)} which can be later specified with concrete scales.

Logical expressions can be used further as classifiers. For instance, an expression «age» is defined as a condition {age > 7} for [End User] profile. When creating the 'child' profiles, we use this expression as a classifier, meaning it will be restricted in two or more distinct sets (e.g. {age = 7-19}) for [Child] and {age > 19}) for [Adult]). It further specialises a [Child] into [Kid] and [Teen] profiles. Each subProfile inherits profile templates and logical expressions that are assigned to their parents.

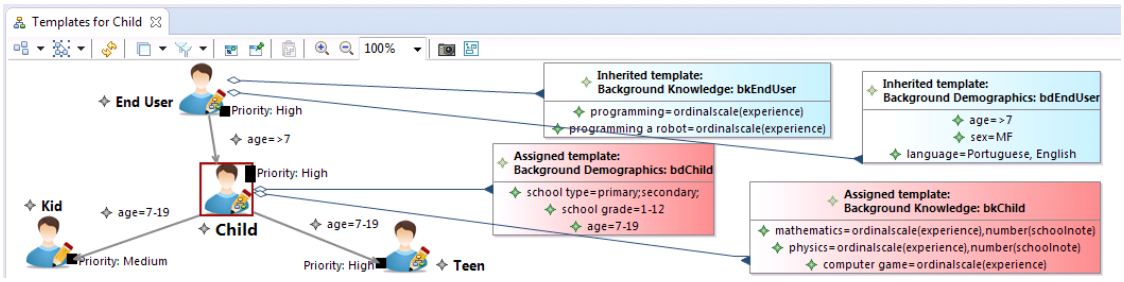


Figure 6.3: User Profile Template (taken from [26])

This kind of structural analysis of domain users for a DSL answers to the question 'Who will use the DSL?'. The next step is to respond the question 'Where will the DSL be used?' by specifying the context environment.

The context environment is described by environmental elements (i.e. CEVariable) such as the social, physical and technical environment (see Figure 6.4). These elements can be reused from architectural descriptions, feature diagrams or other specification files of the language. For each environment element, it is necessary to specify if it is {Mandatory} or {Optional}. Further, it is possible to define contained environment elements and specify their types as a list of values that can be prioritised.

For example, the social environment of Visualino is characterised by [WorkPlace] which is described by «Classroom» «Home», «Public place», «Outdoor». Further, the [Purpose] can be «Competition», «Education» or «Entertainment». Finally, the [Country] is specialised only to «Portugal», as Visualino is being prepared to be distributed only in this country, for now. The physical environment expects from the user to have a

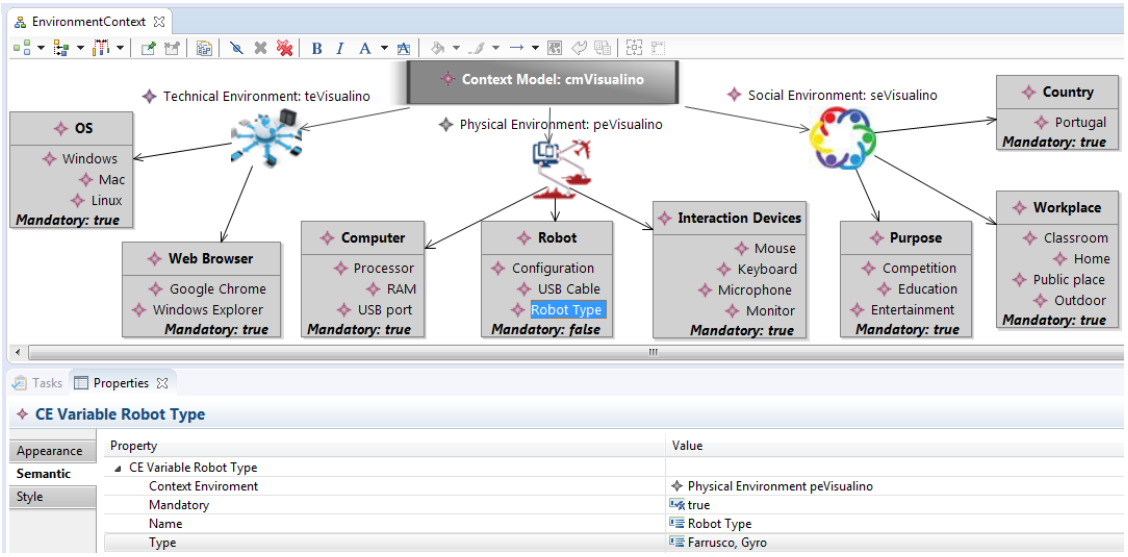


Figure 6.4: Environment Context (taken from [26])

[Computer], that is specified by «Processor», «RAM», and «USB Port», being all mandatory. Also, the physical environment should contain [Interaction Devices], from which «Mouse» and «Keyboard» are mandatory. On the other hand, it is optional, but probably convenient, to have a [Robot] during the use of the Visualino DSL. Each [Robot] can have the specific «Configuration», reflecting the parts of a robot that are configured to be used. A «Robot type» indicates the version of the Arduino robots (i.e. {Farrusco} and {Gyro}), and a «USB Cable». Finally, the technical environment is specified by the [OS] and the [Web Browser], which are both mandatory for running Visualino. The supported [OS] should be «Windows», «Mac» and «Linux». Their concrete versions are stored in a list. Further, the [Web Browser] indicates a «Google Chrome» and «Windows Explorer».

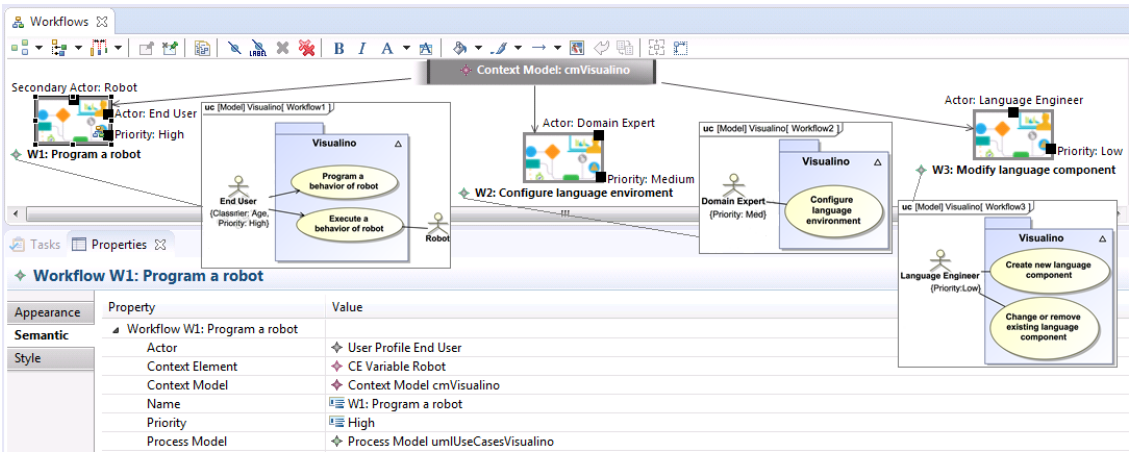


Figure 6.5: Workflows (taken from [26])

The answer to the question 'How is the DSL expected to be used?' can build on scenario-based approaches, such as use case descriptions, which define scenarios and

workflows. (see Figure 6.5). In the case of Visualino, use case specifications are designed with the Sys UML language. These specifications are referred in the «Process Model» instance. The actors who make part of the use case description are represented by different user profiles (users) or environment elements (other systems) that initialise a use case.

A workflow [W1] poses a scenario in which the {End User} (specified as an actor) wants to program a behaviour of a robot and, eventually, execute that behaviour on a physical {Robot} defined in the context environment. In most of the cases, the priority of each workflow can be easily inherited based on the involved user profiles priorities. Following that rule, this case is has a {High} priority. In the workflow [W2] the {Domain Expert} wants to configure the language environment, which inherits the {Medium} priority. Finally, the workflow [W3] represents the case of a {Language Engineer} who wants to create a new language component or change or remove the existing one, in this instance, inheriting the {low} priority.

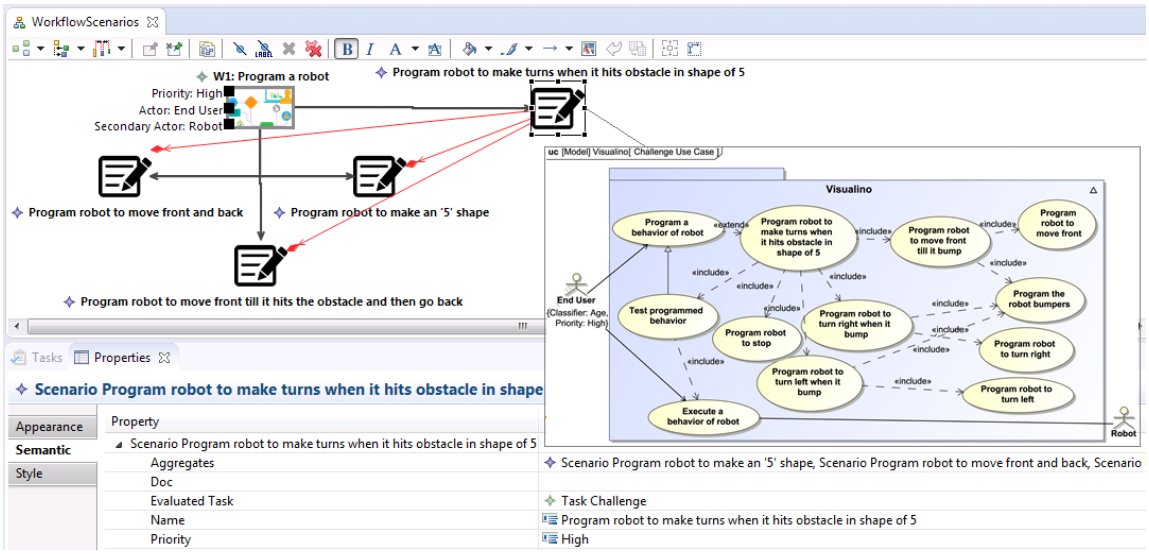


Figure 6.6: Scenarios (taken from [26])

Several scenarios are associated with a workflow [W1] (see Figure 6.6). The first scenario is to program the robot to *move forward* and then *move back*. The second scenario is to program the robot to move along a path similar to a '5'. This scenario includes the following sequence of instructions: *move forward*, *first turn left*, *second turn left*, *first turn right*, *second turn right* and *stop*. In each turn, the robot needs to execute the move operation using the same amount of time and then to make a 90^o angle turn. The third scenario is to program a robot to *move forward* until it *bumps* into some object, then it would *move back* and *stop*. The fourth scenario aggregates the sequences from the previous scenarios. It describes how to program the robot to make a shape of '5', as in the second exercise, but the robot will only turn when it hits the bumper.

6.2.2 Goal model instantiation

In this section, we show how the collected knowledge from the context model can be used in goal analysis (see Figure 6.7). We start with an empty model, as there is no existing goal model from the previous development iteration of Visualino. The 'root' goal represents the highest objective of our analysis, that is to achieve [Quality in Use]. It is prioritised as high, indicating that usability evaluation is highly necessary for this development project. Its «scope» is set to be applicable on {all} CM, designed in the previous section. The «actor» responsible for achievement of the highest usability goal reflects all stakeholders included in the development. The {Children robotics expert} and {Visualino development} represent the domain experts and language engineers from Artica, while {Language Evaluator} represents the authors that were included as expert evaluators.

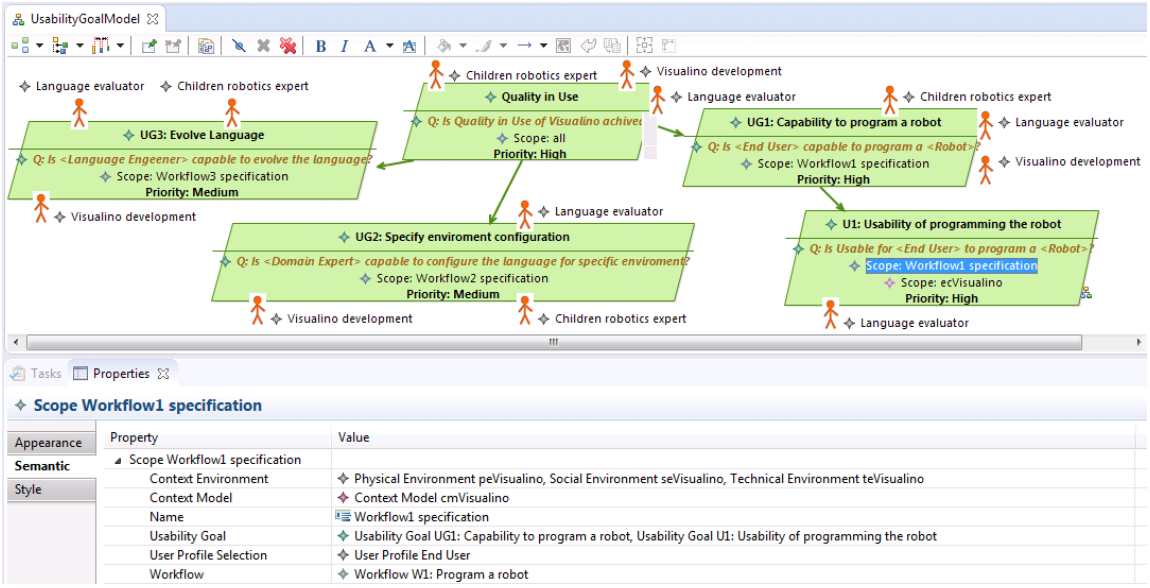


Figure 6.7: Usability Goal Model (taken from [26])

The [Quality in Use] is divided into three «subGoals», namely [UG1], [UG2] and [UG3], which reflect different workflows from Figure 6.5. The «scope» objects, associated with these goals, restrict a CM in a way that they reflect parts for which each workflow applies. They are only associated to a sub-branch of the user hierarchy model that consider a user profile of the workflow's actor. A goal [UG1] is given the high «priority» as its «scope» reflects the {End User} profile and {Workflow1} workflow. As it is suggested by goal modelling approaches, such as KAOS or i*, we further divide this goal into the subgoal [U1] for which the evaluation the responsible is just one «actor», {Language evaluator}.

A usability goal [U1] specifies that the «actor» {Language evaluator} evaluates the [Usability to program a robot] and represents a quality goal usually measured by non-functional requirements (see Figure 6.8). It answers the «question» {Is it usable for an

[End User] to program a robot?}. Further, a measurable method is specified as [Programming a <Use Case> is usable] and is related to particular usability requirements. This method is related to a functional goal [F1]. A functional goal specifies that the {Visualino development} verifies that it is possible to program a robot. It answers the «question» {Which scenarios are verified to be programmable using Visualino?} by testing requirements which describe the necessary functionalities to execute the selected «Test case», {Workflow1}.

Further, usability requirements are specified in a way they can be measured in the current phase of development. In the case of Visualino, we focus on the evaluation activity which was planned to be executed after the implementation. In this phase, the following measurable requirements are found to impact the goal [U1]: [Effectiveness], measured by a {Correctness of programmed <Use Case>.}; [Satisfaction], measured by {Satisfaction questions related to <Use Case>.}; [Efficiency], measured by {Time to program the <Use Case>.}; and, [Learnability], measured by {Success to program the learned Use Case.}. Note that other measures could be appropriate in different development stages: for the design phase without working prototype, readability and understandability [23], or for a proof of concept, feasibility and integrability. The catalogue of usability metrics (Section 5.8) is expected to support this choice and document the metrics from existing experiences.

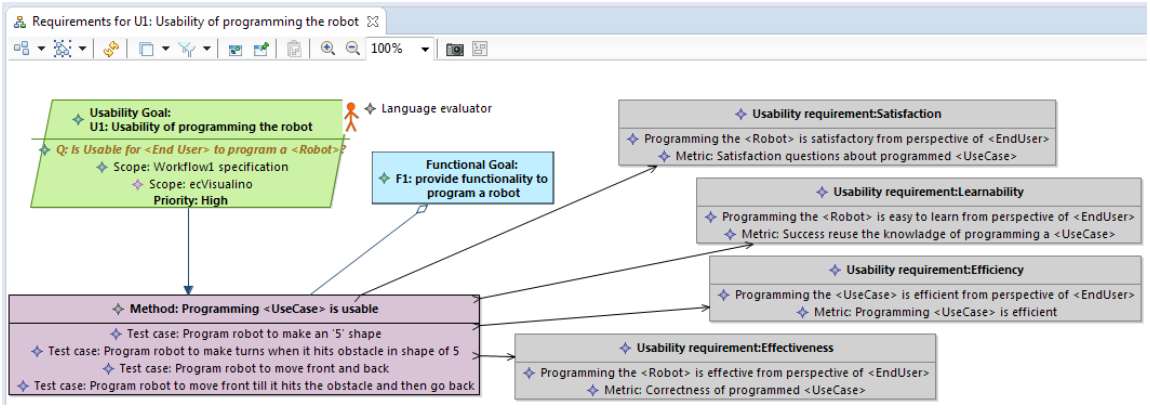


Figure 6.8: Usability requirements for usability goal [U1] (taken from [26])

6.2.3 Evaluation model instantiation

As mentioned previously, language usability can be promoted by combining Usability Engineering techniques with ESE. In this section, we present an evaluation model for the empirical study of our target DSL, Visualino, which is specified based on its [CM] and [GM]. The first thing to do is to model the evaluation objectives (see Figure 6.9). The primary language of the presented study is a [Visualino]. It was found useful to compare its early prototype to other widely used language for programming robots, namely [Lego], representing a second language in [EM]. The participants were expected to match the

{Teen} «profile», as the experiment was performed with secondary school subjects during the ExpoFCT event[8] at Universidade Nova de Lisboa in Portugal. The goal was to evaluate the «usability goal» {U1 - Usability of programming the robot}, specified by GM. Namely, each evaluation goal was to validate «usability requirement»: first addressing the {Effectiveness}, and second by {Satisfaction}. For each, we have specified the research question in the form of a «problem» using the GQM template, from which we have derived the «hypothesis».

We analysed the Lego context by giving attention to the workflows and environment that are comparable to the Visualino's, and the ones that are in the scope of the «usability goal» which we are evaluating. During this analysis, it was possible to identify the context in which the evaluation study was appropriate to be performed - this is described by the evaluation context (see Figure 6.10). The evaluation context is defined by: «user profile» of [ExpoFCTParticipant], which is {Teen}; the «context environment» that represents where the experiment will take place; and, by the «workflow» that will be used. For instance, for the social environment, the «country» was set to {Portugal}, and the «workPlace» was selected to be a {Classroom}, while the «purpose» was chosen to be a {Competition} to create a motivating environment for the participants. The technical environment was instantiated by «OS» {Windows 7} and the «browser» {Google Chrome}. The physical environment details the information about the «computers», «interaction devices» and «robots» that will be used. The preferred «configuration» for both robots (Lego and Farrusco) is specified, to assure that the robots will have the same functionality. We can note how the environmental elements of Visualino's CM are initialized to concrete values, which are saved to value lists associated with this objects. Finally, the observed <workflow> is the {Workflow1}, which includes all scenarios defined previously, as they are executable in this specific context.

We used a 'between groups' design where participants were randomly assigned to either programming a robot using [Visualino] or [Lego], but not both. The evaluation process [emVisualinoProcess] was designed to start with a 30 minutes learning session, during which the participants should learn the language's concepts and how to solve three basic exercises that reflect the first three scenarios specified in {Workflow1}. These three activities and the tool help were prepared as printable documents, presentations and video demos with the version of the language under evaluation. The materials were saved as documentation objects. Further, it is necessary to define a «test model» through survey modelling for {Background Questionnaire}, which was filled in during the evaluation session. The evaluation continued with 15 minutes of a competition session during which the participants tried to solve a programming {Challenge} reflecting the fourth scenario, which was aggregating the sequences from the previous scenarios. The «test model» for this session was specified with the interaction model, as we wanted to capture relevant events by recording the contents of the computer screen during the session. Finally,

[8] https://www.expo.fct.unl.pt/ (accessed September 19, 2017)

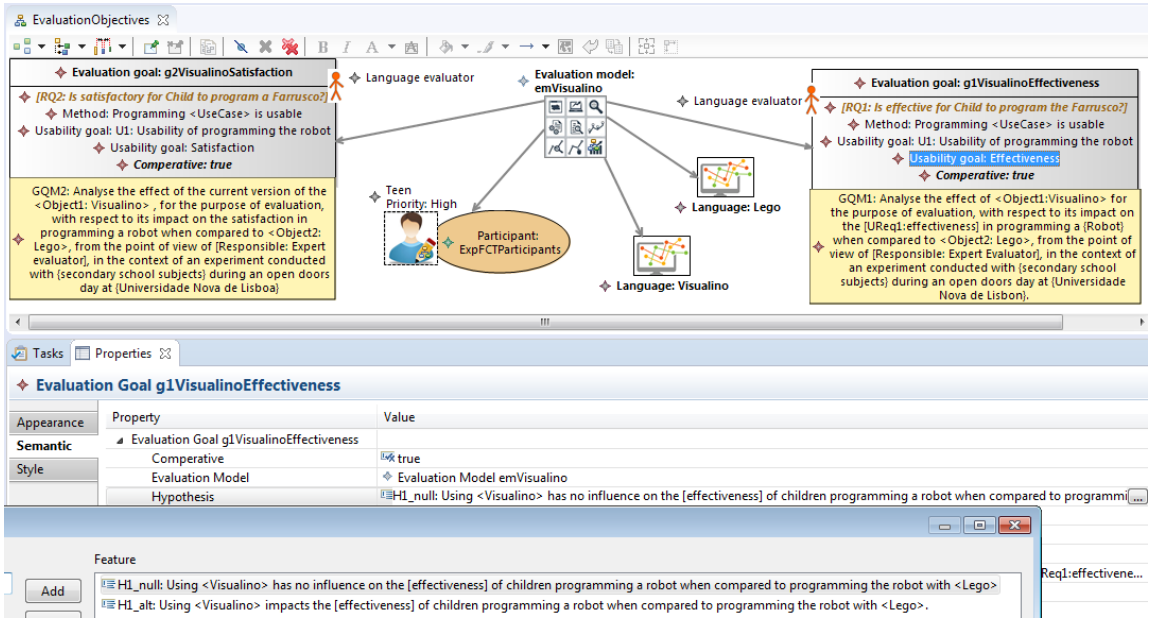


Figure 6.9: Evaluation objectives (taken from [26])

there was a feedback session, taking up to 5 minutes, to collect the children's subjective opinions about using the language they were assigned to, by answering to the {Satisfaction Questionnaire}, which was modelled by the survey modelling activity, but dependent on the specification of the interaction model.

While preparing this evaluation, from the teaching materials and the events analysis, we were also creating the documentation of Visualino and its application scenarios that can be shipped along with the software product, as they were reviewed and improved for the purposes of the experiment. This material, referring to a specific scenario can be documented within the CM. Further, we illustrate how the test models are designed using the interaction modelling and the survey modelling activities.

6.2.4 Interaction model instantiation

In this section, we instantiate the interaction test model (see Figure 6.11), which is used during the competition session of evaluation. This model encapsulates the «interaction syntax» from both, Visualino and Lego. Further, it is specified for the «task» which is associated with the concrete scenario from the evaluation context, named {Challenge}. A correct solution of the problem was provided by the Artica developers for Visualino, and by a language engineer who had experience with Lego.

By analysing the given task and the evaluation model, three events are defined to be captured. To measure «usability requirement» {Effectiveness}, described as correctness of solving <Use Case:Challenge>, [Event1] and [Event3] are specified. [Event1] captures Success (S) or Fail (F) of modelling the following concepts: Bumper, First Turn Left, Second Turn Left, First Turn Right, and Second Turn Right. It is captured manually observing

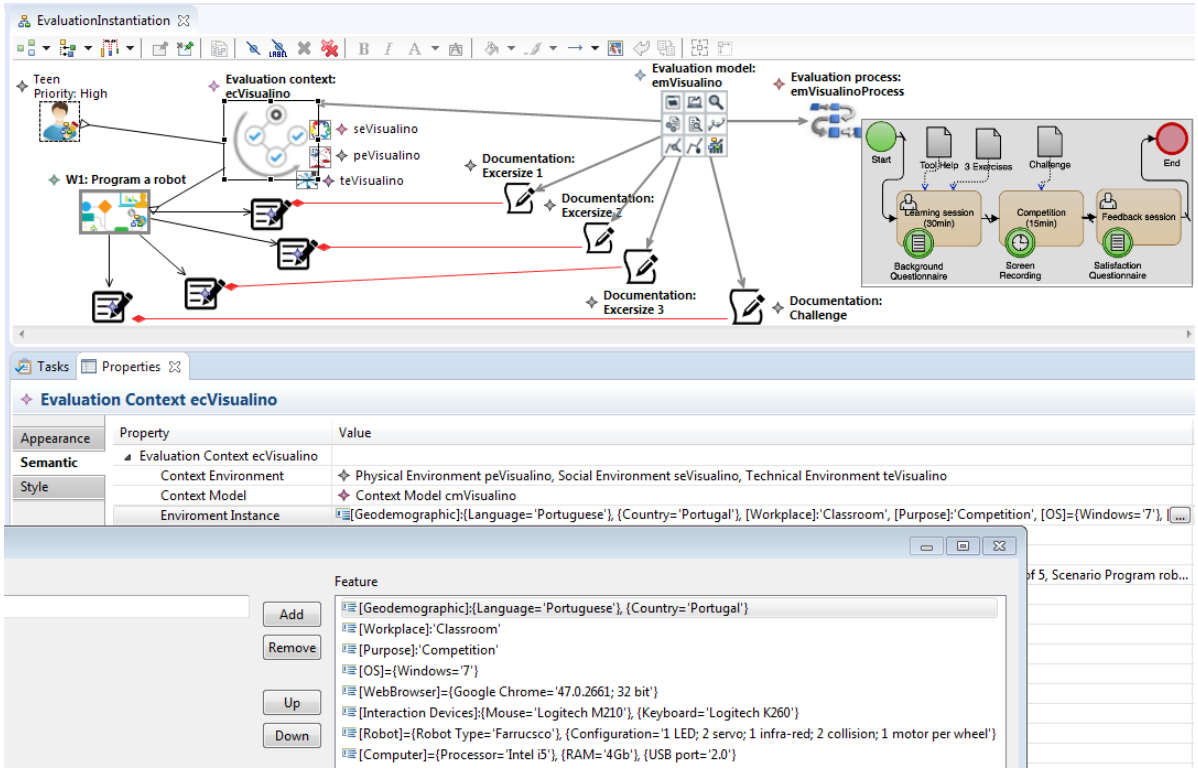


Figure 6.10: Evaluation Instantiation (taken from [26])

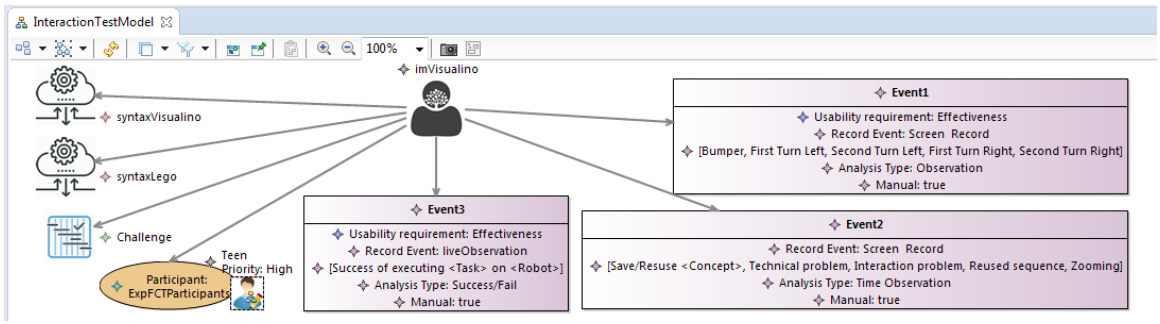


Figure 6.11: Interaction Test Model (taken from [26])

{Screen Record}, e.g. videos of the participant's interaction with the language while trying to provide a correct solution. Further, [Event3] was marked with Success (S) if the robot successfully performed the challenge in the arena; Fail (F) if the team tried until the time limit, but did not succeed to program the robot correctly; and Quit (Q), if the team gave up. Finally, in the [Event3], we assessed additional useful information during the video analysis, like if the team: (i) saved or reused the previous exercises; (ii) experienced technical problems or functional errors; (iii) had interaction difficulties (e.g. using copy/paste for visual objects or connecting concepts in sequence); (iv) reused previously constructed sequences (within the same exercise); or (v) used any other additional language features (e.g. zooming). [Event3] registers the time of occurrence of certain <captureEvents>, measured from the beginning of the competition session and can serve as a source for

ideas on improvement of certain language features for language engineers.

The interaction results are analysed by the expert evaluator, who prepared the data collection sheets and has put forward the analysis. The correlation analysis was supported by the SPSS tool [131], and the graphs were generated in Excel.

6.2.5 Survey model instantiation

By performing survey modelling, we instantiated the test models (see Figure 6.12) for the background questionnaire used during the learning session, and for the feedback questionnaire used during the feedback session. The survey forms were composed of Smileyometers, which are found to be appropriate for children questionnaires [181]. While answering to forms, children were assisted by an adult (one of the experiment assistants). This was done to ensure that there were no misinterpretations of questions and answers, and to confirm that participants did not experience reading problems. As participants were grouped into teams, the participants' individual answers to questionnaires were merged, and the mean rate was computed within each team, for each answer.

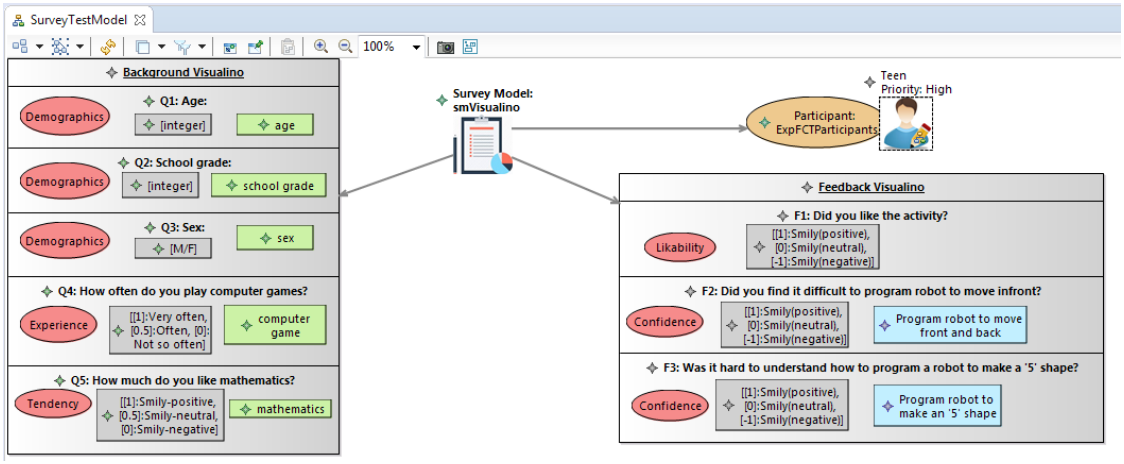


Figure 6.12: Survey Test Model (taken from [26])

The background questionnaire [BackgroundVisualino] is related with the {Teen} «profile». Each question in this questionnaire is related to one of the logical expressions associated with evaluated profile in CM. For instance, for [Q1:Age], the «logical expression» {age} was associated. In this case, the age is defined as an actual value ranging from 13-19. As so, the expected answer is defined as an {integer} ranging from 13 to 19. If another value occurs, the participant's data is discarded from further analysis. The collected data includes «indicators» defined as {Demographics}. These help to verify if there are other factors, such as age or gender, influencing the results. On the other hand, certain «indicators» like {Experience} were calculated as a average value of this group of questions to serve for further impact analysis. Also, the abstract scale, for instance of «logical expression» {Computer game} was instantiated as 3-point Likert scale {[1]:

Very often, [0.5] Often, [0] Not so often} with concrete weights which are used for later calculations.

The feedback survey was designed to measure the «requirement» {Satisfaction}, which was characterised by the following «indicators»: {Confidence}, reflecting how confident children were about their solution, {Likability}, reflecting how interesting and enjoyable they found the challenge itself; and {Learnability}, reflecting how useful they found what was taught during the learning session to help them facing the final challenge. Each question can be related to a concrete «scenario» included in EM. For instance, the question [F2: Did you find it difficult to program the robot to move in front?] indicates {Confidence} of the participant regarding the learned «scenario» {Program robot to move front and back}. The «scale» for all satisfaction questions reflects positive or negative experiences and is defined as {[1]: Smiley[positive], [0]: Smiley[neutral], [-1]: Smiley[negative]}.

6.2.6 Result model instantiation

To evaluate the usability of {Visualino} and {Lego} we reused techniques of UI evaluation that imply the involvement of real users as subjects of controlled experiments. A comparison criterion was based on the correctness of the problem solution, the time to solve it and personal preference. To get correct interpretations of our results, it was mandatory to profile the users. The criteria that were observed were their age, gender, and programming background. Further, we collected the participant's feedback and success rate regarding their experience with a language. During the report modelling activity, we declared all of these results in a Report Model [expo2015result] (see Figure 6.13). The results specified by the interaction and survey models are documented for each event or questionnaire. They are correlated to evaluation results, whose objective was to analyse the {Effectiveness} and {Satisfaction} «requirements» for each language. The statistical analysis documents are assigned as «outsideRef» properties.

The observed evaluation results of the second iteration of the design of the DSL showed convergence and highlighted new possible improvements of Visualino. The children subjects were still having problems in making a right solution, having a small {Effectiveness} score of 0.37 for Visualino. The subjects were expressing likability toward Lego, while toward a Visualino they were indifferent. Based on these results, we created the recommended GM [Recommendations] consisting of new requirements impacting the usability goal for [U1] and functional goal [F1].

6.3 Pilot evaluation study

In this section, we report on the pilot assessment of the feasibility of the USE-ME tool with master students in computer science which were involved in a DSL course. In total, four groups were consisting of two or three participants which were developing the following DSLs:

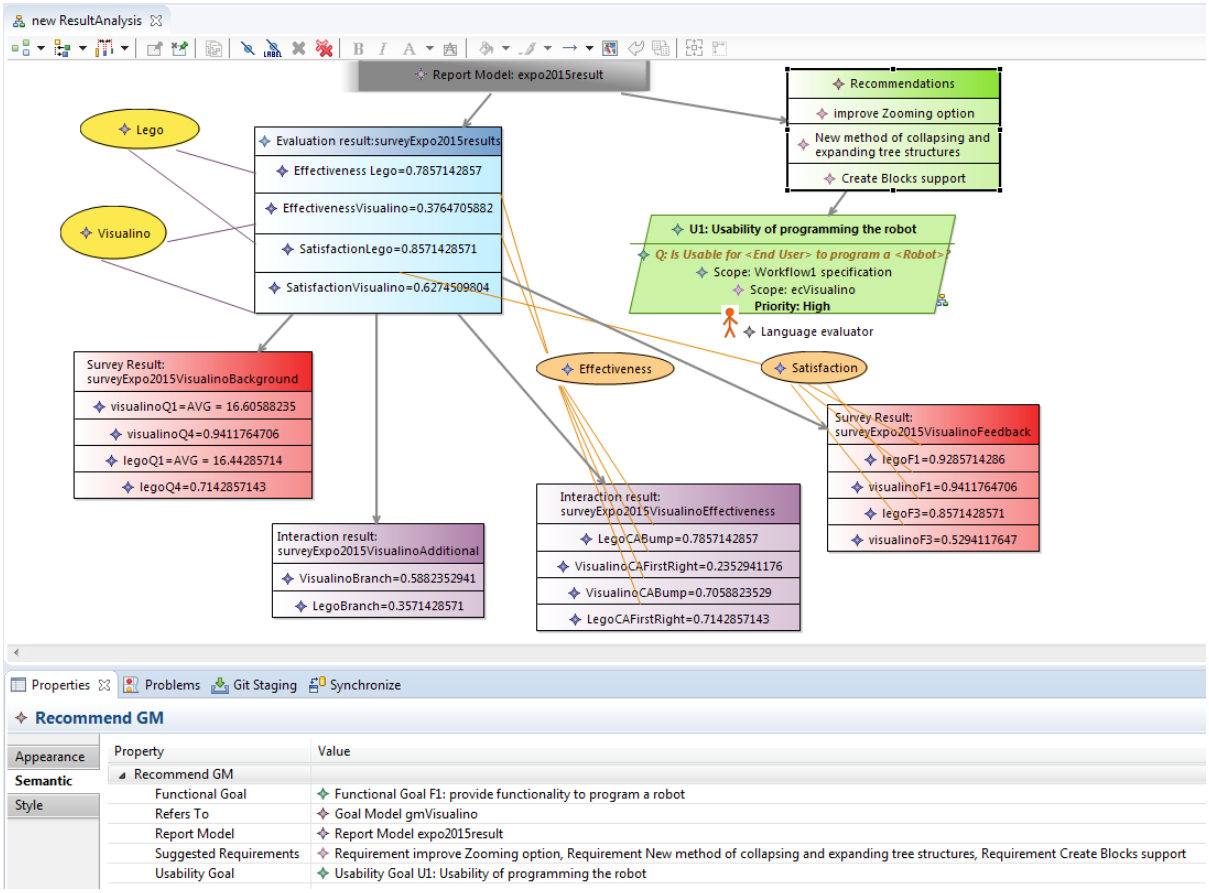


Figure 6.13: Report Model (taken from [26])

- *DSL Spreadsheets* - a DSL that transforms an activity graph into a Gantt chart. The target users of this DSL are project managers.
- *Gestures Kinect* - a DSL which supports specification of communication between navy users using a Kinect device. The target users of this DSL are navy operators.
- *Peddy Paper* - a DSL which creates several instances of personalised rally paper scripts. The target users are rally paper organisers.
- *Smart House* - a DSL which supports a design of the elements and operations for a smart home. It is meant to be used by house owners.

The evaluation sessions were prepared as shown in Figure 6.14. The first learning session took place after four weeks of the DSL development. Students were introduced to the usability evaluation during a 2h theoretical lecture. For the next two hours, the students received a tutorial on the USE-ME tool and were guided to perform installation and set up working environment. Also, the students were given a participation questionnaire to fill in and describe a purpose of their DSL. At the end of the session, they were given the background questionnaire to fill in. The following two weekly four-hour sessions consisted of USE-ME hands on labs. The students were introduced to the modelling

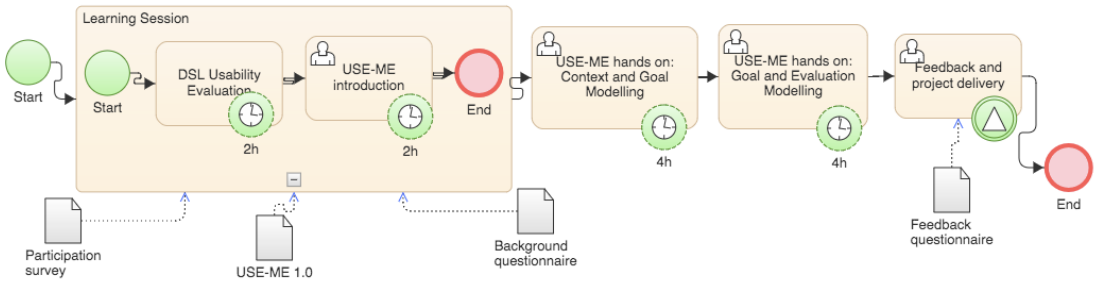


Figure 6.14: Pilot session process Model (taken from [26])

activities followed by Visualino example. For each activity, the same student from the group was using the tool to create USE-ME models for their DSL, while other students from the group were helping in deciding what would be a correct specification. Finally, students were asked to try to finish the learned models and deliver them as a part of the DSL course. After delivery, the students who were using the tool to model were asked to fill in a feedback questionnaire.

The evaluation deliveries (project reports, USE-ME models), background and feedback questionnaire results and evaluation results can be found at [28]. Here we shortly describe the obtained results:

The participants were master students with intermediate knowledge of modelling techniques, namely having a background knowledge of UML (class diagrams, use cases, activity diagrams and interaction diagrams). Also, participants knowledge related to the working environment was advanced. However, the participants had little or no knowledge regarding HCI, especially empirical experiments or the usability testing. As the objectives of the USE-ME conceptual framework are to support Language Engineers in specifying the usability evaluations, as the Expert Evaluators are often found to be too expensive to be included, we find that participants are adequate surrogates for the target end user of {Language Engineer} profile which is to be supported by USE-ME conceptual framework.

Table 6.1 presents the results of the USE-ME model validation for each reported DSL. We have analysed all diagrams which should be provided for Context Modelling, Goal Modelling and Evaluation Modelling activities. The diagrams were graded regarding their completeness and correctness as follows: (0) - no delivered model, (0.2) - very low, (0.4) - low , (0.6) - satisfactory, (0.8) - good, (1) - very good. We can note that all groups expect the Spreadsheet DSL manage at least satisfactory to specify their models. However, Spreadsheet group participated only for a two hours of a hands-on session and this impacted the low score of their reports. In each case, we can observe that delivered models were satisfactory and by this indicating the understandability of the conceptual framework in specifying the usability evaluation within a small amount of time. However, it seems the biggest challenge for participants was to figure out how to specify usability goal model and associated requirements. In its current version, the prototype does not

Table 6.1: Validation table of USE-ME models produced for different DSLs

DSL	***Spread Sheets***	***Gestures Kinect***	***Peddy Paper***	***Smart House***	**AVG**
Participation time	6h	12h	12h	12h	
CONTEXT MODELING	***0.36***	***0.72***	***0.56***	***0.76***	**0.6**
User Hierarchy	0.6	0.8	0.8	1	0.8
User Profile Template	0.4	0.6	0.4	0.6	0.5
Environment Context	0.2	0.8	0.6	0.8	0.6
Workflows	0.2	0.8	0.2	0.8	0.5
Scenarios	0.4	0.6	0.8	0.6	0.6
GOAL MODELING	***0***	***1***	***0.4***	***0.5***	**0.475**
Usability Goal Model	0	1	0.4	0.8	0.55
Usability Requirements	0	1	0.4	0.2	0.4
EVALUATION MODELING	***0.2***	***0.6***	***0.7***	***0.7***	**0.55**
Evaluation objectives	0.2	0.8	0.6	0.6	0.55
Evaluation instantiation	0.2	0.4	0.8	0.6	0.5
Interaction Test Model	0.2	0.4	0.8	0.8	0.55
Survey Test Model	0.2	0.8	0.6	0.8	0.6

yet support the usability catalogue. We believe that this extension will help users without much knowledge about requirement engineering and quality evaluation to define properly usability objectives for their case.

The participants from the groups which were having a bit higher understanding of HCI were naturally selected to be responsible for modelling with the USE-ME conceptual framework. That is why just these participants were asked to provide a feedback about the tool. Participants authorised the publication of project results to Zenodo library. Although they found usability evaluations necessary for a DSL deployment in practice, they did not find the modelling activity interesting. One of the main reason was that the modelled evaluation was not to be executed, making it less interesting for their project purpose. The Gesture Kinect group found that it was easy to understand the USE-ME conceptual framework. The Peddy Paper group found it on another hand not to be so easy, pointing out there were too many steps to follow. The Smart House and Spreadsheets groups found conceptual framework more or less easy, pointing the lack of a guided cycle and highlighting a usefulness of the Visualino example for understanding an conceptual framework. All of them reported to be able to discover environmental elements which they did not consider before in development, and found easy to create a user hierarchy and more or less easy to specify other context model elements. However, they found the modelling of usability goals they found a bit harder but still manageable. Two groups found useful to reuse context model elements in goal specifications, or during the evaluation modelling. They found it to be easy to creating test model specifications for evaluation.

In general, they did not feel very confident while using a USE-ME tool, however, they

find a tool expressive enough for purpose of specifying usability evaluation, and also suitable for another kind of software products. However, none of the participants is familiar with any other tool which supports usability evaluation.

Chapter 7

Applicability of the USE-ME conceptual framework

In this Chapter, we establish the relation of the USE-ME process we proposed in Chapter 5 with the DSL development cycle presented in Section 2.1.

7.1 Integration of the USE-ME conceptual framework with DSL development phases

In Figure 7.1 we present the integration of the development process where one cycle of USE-ME process is performed during one development iteration. In this Figure, we show how to relate one complete cycle of the USE-ME activities with each of the development phases.

During the **Decision** and **Domain Analysis** phases of the DSL development, we perform together the *Context Modelling* and *Goal Modelling* USE-ME activities. This way, the activities can share information which is being specified and discussed in both phases while language engineers and domain experts produce the relevant DSL's modelling artefacts. For instance, the feature diagram brings useful information about environmental elements of the [Context Model], and the [Goal Model] can provide useful hints to discover usability goals and requirements, establishing a traceability link to the existing goal model produced in the **Domain Analysis**. Further, during the **DSL Design** phase the *Goal Modelling* activity should be concluded and, therefore, the *Evaluation Modelling* activity can start. The evaluation is dependent on the scope of the DSL modelling that should now be implemented. For instance, at an early stage, it is already possible to define the objective of the evaluation study, its process, context and discuss if the comparative evaluation should take place, or not, besides presenting the alternatives to be considered.

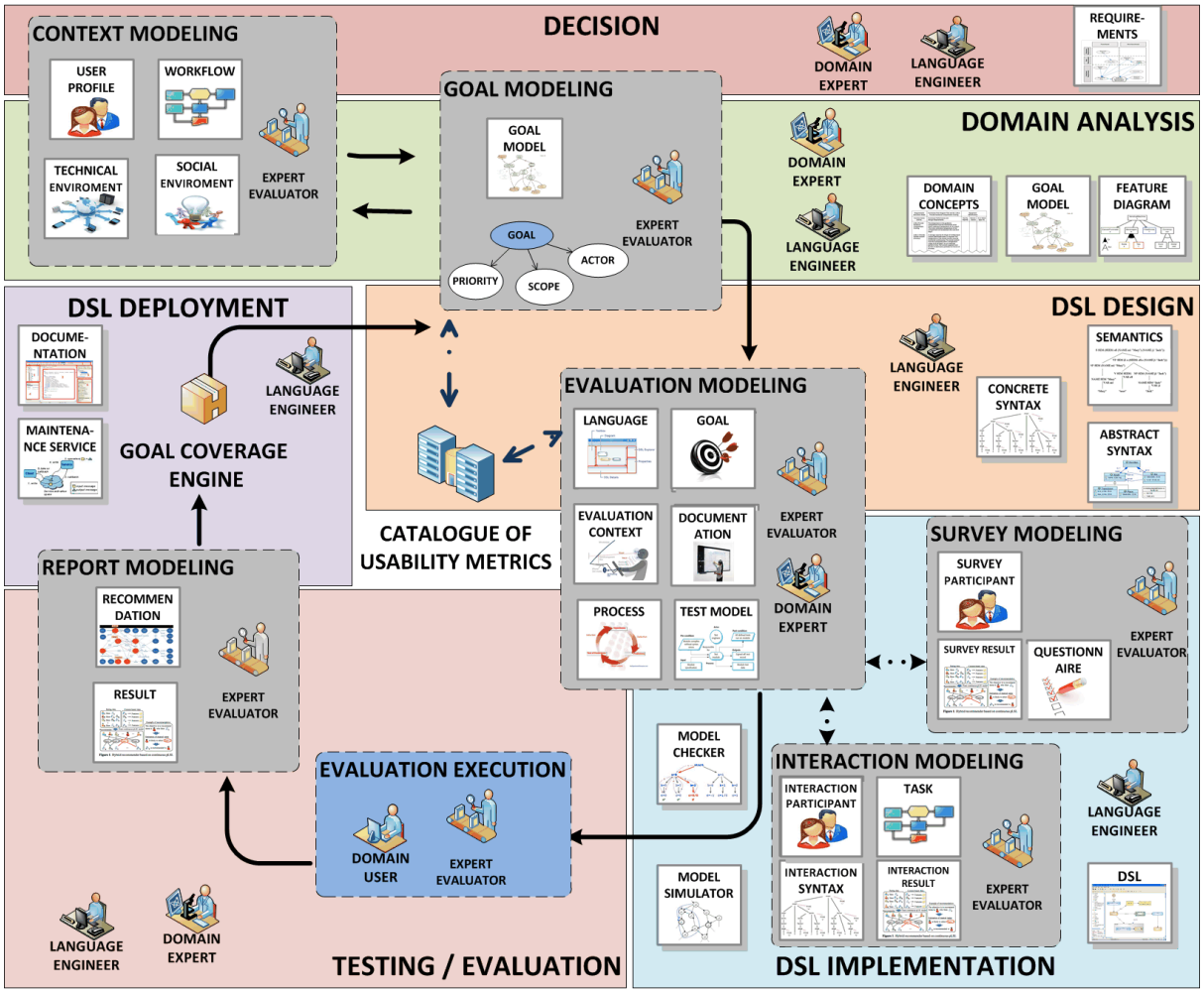


Figure 7.1: Integration of USE-ME conceptual framework with DSL development phases (taken from [26])

During the **DSL Implementation** phase, test models are created for the evaluation activity thanks to the *Survey Modelling* and *Interaction Modelling* activities. Further, in the **Testing/Evaluation** phase of DSL development, the *Evaluation Execution* takes place in the USE-ME process. Usability goal measures can depend on the provided functionalities of the DSLs (e.g. ones needed for carrying out chosen test scenario). Therefore, these required functionalities should pass functional tests and the evaluation process should reflect the dependencies with this functional tests. For instance, the usability evaluation execution can turn to be unsuccessful, if the participants fail to perform the evaluation tasks due to technical problems. Finally, the *Report Modelling* activity can be performed and, if necessary, extended to the **DSL deployment** phase. Finally, the [Report Model] is used to update the previous [Goal Model] and to make a decision about the next development cycle.

However we have described a possible application of the USE-ME's process that encompasses the full iteration of the DSL development life-cycle, sometimes the complete

USE-ME cycle can be used to cover a single DSL development phase. For instance, during the **Domain Analysis** phase in which the requirements and objectives are not yet clear, we can model the survey which will help us to clarify the objectives from the different stakeholders. In this case, the resulting goal or context model is justified. On the other hand, during the **DSL Design** phase, we might want to experiment with different syntax designs, and it is useful to create an assessment to evaluate their readability and understandability. Further, during an **DSL Implementation** phase, various prototypes can be implemented and evaluated before proceeding with the **DSL Deployment**. Finally, during the **Testing/Evaluation** phase, we might want to carry out several interdependent evaluations, e.g. previous ones impacting the extension of the [Context Model] and [Goal Model] activities.

7.2 Application of USE-ME to incremental iterative DSL development

The USE-ME conceptual framework reuses the specifications along different iterations so that it is not necessary to start again from scratch with the analysis and design in each iteration. When introducing small extensions or changes to the DSLs' syntax or semantics, it will be necessary to extend the existing USE-ME models with 'new' context instances (e.g. new/extended user type, environment or user story). The updated evaluation goals should take into consideration the new features and/or expected usability improvements over a previous version of the DSL. It is useful to perform the comparative evaluation with a previous version of the DSL, to observe if an improvement happened. For instance, in the case of Visualino (see Section 9.4), we reused the experimental design, evaluation process and their instruments (surveys and an interaction test model) from the previous iteration. This gave us means to comparatively analyse the results from the first and second iteration. To support the organisation and planning of the iterative development, and increase the transparency of the development tasks, we can benefit from existing management tools support used for agile approaches (for instance we applied Scrum and Pivotal Tracker during the FlowSL case study (see Section 9.3).

To trace changes in a DSL, we can benefit from existing incremental language development approaches such as LISA [147]. In this work, when the language is extended with new features, this corresponds to a new language containing specifications of the change together with the description of the previous language. For example, Mernik et.al [147] extend (in the sense of object-oriented extension) a simple language describing a robot movement with a new language which also calculates when the robot will reach the final position. For performing this extension, there should be a concrete motivation behind. For instance, in this case, new scenarios should be added to the previous workflow in a Context Model. This change does not imply the addition of the new Usability goal, but rather an extension of the existing ones with a new scenario. The evaluation design can

be kept and applied to a refined scenario. On another case, also taken from [147], the authors perform an extension which supports the robot for cleaning. This change refines the previous end user profile (the user wanted 'to use a robot'), to a new profile where the end user wants 'to use a robot *for cleaning*'. Also, the environment context is specified more in depth reflecting the robots for cleaning and environment where they are used. Finally, the new user stories may have different actors and will be documented under the new workflow. The evaluation should redefine the usability goals, and probably create new ones which take into consideration the new context.

Early evaluations and extensions without a significant change of context or usability goals, can be performed with domain experts, while other users should be involved in evaluations later on. By performing early assessments with more available users the evaluation design and its metrics are being validated, and a risk of not obtaining a return of the investment in extensive evaluations is lowered. Also, as mentioned previously, bigger investment into usability evaluations should be applied to the DSLs which target a wide scope of the end users, especially those who are not exactly reflected by the domain experts profile which is included in the development. For the 'in house', or small DSLs developed for a very small set of users, especially those included in its development process, an application of our systematic approach as a whole might be too expensive, but the analysis procedure can still be found useful.

7.3 Applicability of the conceptual framework outside of the scope of model-based DSLs under development

7.3.1 Previously released DSLs

We followed the USE-ME conceptual framework on two industrial DSLs. In both case studies, we were not directly involved in the development of the DSLs but participated as evaluation experts. These DSLs were developed from scratch: an internal DSL based on Ruby [23]; an external DSL described in Section 9.4. We chose these case studies as we did not find many examples of usability evaluations involving end users early in the DSLs' development cycle (see Chapter 3). If we think of DSLs already existing in the market, the USE-ME process will be similar to the process used with DSLs which are under development. However, with previously released DSLs, it should already be clear **Who** are the users, **What** are the user stories and **Where** the DSL is being used. It is expected that it will be possible to reuse most of this knowledge from already existing artefacts and specify the usability goals and metrics in an easier fashion. During the evaluation modelling, the assessments can be planned to be performed automatically (e.g. with automated usage data collection) and remotely. With a large number of regular users, more data can be obtained.

On the other hand, if there is still information missing to create a complete context or

goal model, it is still possible to plan the evaluation, by sticking to the information that exists. For instance, if there is no formal specification/documentation of the workflows, this information can be omitted. However, the evaluation model will be designed accordingly, e.g. to evaluate the satisfaction with a language or the readability of design concepts. It is also possible to create an interaction model which captures just 'random' roll-back cycles (semantic errors) during a common use of a product (e.g. placing and deleting certain commands repetitively in a certain sequence can help identifying misinterpretation issues of a concrete syntax element).

7.3.2 Grammar-based DSLs

Both of our case studies were developed by using meta-modelling tools (MetaEdit and Eclipse EMF, respectively). Our process is not restricted to model-based DSLs and could, in principle, be applicable also to grammar-based DSLs or even to GPLs. However, this would require adapting our prototype to cope with their architecture. This would be feasible but is beyond the scope of this thesis. It would be necessary to apply modelling techniques (e.g. reverse engineering), to create certain artefacts (e.g. goal model, UML diagrams, etc.).

7.4 Taking the role of Expert Evaluator during the DSL development cycle

The modelling activities in our conceptual framework are presented as activities of an Expert Evaluator which, in practice, is not typically included in the DSL development process. We mentioned in Section 2.2 that this HCI expert usually implements complex experimental usability evaluation studies. The HCI expert profile includes comprehension of user profiling, experimental approaches and usability evaluation methods. However, when applying the proposed systematic approach, these kinds of experts may lack the knowledge about model-driven methods and requirements engineering, and it may be not trivial for them to grasp the modelling concepts and tools. Therefore, these experts should be introduced to abstraction modelling and trained to use the modelling support with a focus on mastering the modelling concepts present at the USE-ME conceptual framework (e.g. goal modelling, process modelling, UML modelling).

Moreover, we found that in case there is no person included in the development team with evaluation expertise, the knowledge of these experts can be transmitted to a typical DSL Engineer through the USE-ME conceptual framework. The conceptual framework was validated by researchers from NOVA-LINCS research centre, who are experts in MDD and DSL development and were not involved in the conceptual framework development. Some of them are also knowledgeable about the experimental software engineering and requirements engineering. These experts provided valuable feedback and improvement

suggestions over the conceptual framework. However, people with high level of experience might also not be available during the DSL development process to take the role of Expert Evaluators. Inexperienced engineers can still be suitable for the Expert Evaluator role in the DSL development project. As a proof of concept, we performed a preliminary pilot evaluation of the conceptual framework prototype with master students in informatics who had knowledge about MDD and DSL development [28]. However, these engineers should be trained in user profiling, usability evaluation methods, experimental software engineering and requirements engineering practices.

Chapter 8

USE-ME evaluation

We used experts feedback as a form of evaluating USE-ME, concerning its usefulness. This evaluation was itself planned using USE-ME, to express the context and usability objectives of the USE-ME support (introduced in chapter 6), as well as to model the expert evaluation and present results. The model instantiation (*pt.fct.unl.novalincs.useme.example.UseMe*) can be found in USE-ME GitHub repository: github.com/akki55/useme/tree/master/examples/.

8.1 USE-ME context and goal model

In this Section, we specify the context of use which we considered while building a USE-ME tool support and which justifies our design decisions.

8.1.1 User hierarchy and user profiles

First, we defined the user profiles which are considered to use the USE-ME conceptual framework.

Figure 8.1 presents the user hierarchy, for which we define several User profiles. We characterized any USE-ME Stakeholder with a Profile template reflecting demographics, which contains the following classifiers:

- Age - a factor which can indicate if there are users of certain age groups which can adapt systematic approach easier. However, the conceptual framework is not considering the stakeholders which are children (under 18 years).

- Country - factor which can influence the user adoption of the conceptual framework with properties of certain country (e.g. education system, accessibility of information, necessity of adoption of the USE-ME conceptual framework)

- Institution - factor which can influence the user adoption of the conceptual framework with properties of the certain institution
- Degree - the level of education moderates the ability to deal with the complexity of the conceptual framework
- Experience background - allows assessing the extent to which people coming from different contexts (e.g. academic (i.e. research) or industrial (i.e. practical)) influences their perception and objectives while assessing USE-ME.
- E-Mail - person's identifier/contact
- Name - person's secondary identifier
- Personal Page - person's personal web page
- Role - role which person is taking in relation to USE-ME development

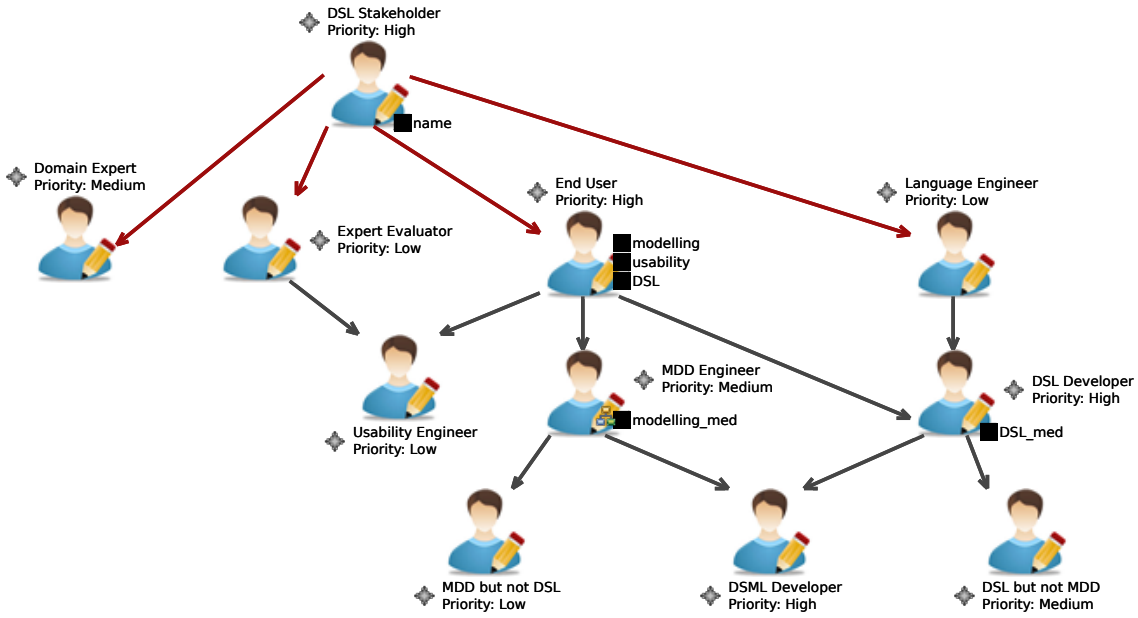


Figure 8.1: USE-ME User Hierarchy diagram

The thesis candidate played a double role in USE-ME development, as a **Language Engineer** and **Expert Evaluator**, while the supervisors of the thesis played the role of **Domain Experts**. The **End User** profile represents people not involved in USE-ME development, but who are potential users of USE-ME, as software language engineers. It is classified into a three sub-profiles, in regard to their knowledge about modelling, usability or DSLs and characterized by a profile template which reflects the relevant knowledge about:

- DSL - as the USE-ME conceptual framework is intended for DSL development

- Modelling - as the USE-ME conceptual framework is developed using MDD approach
- Usability - as the USE-ME conceptual framework is developed to support usability evaluation
- UML - as the USE-ME conceptual framework is specified by UML diagrams
- Requirement Engineering - as the USE-ME conceptual framework is supporting the validation of usability goals and requirements (e.g. categorized as non-functional in requirement engineering), and therefore is based on requirement engineering goal modelling practice
- Agile Development - as the USE-ME conceptual framework is meant to be applied iteratively and incrementally
- HCI - as the usability evaluations are meant to improve HCI between a user (i.e End User) and a software product (i.e. DSL).

The **DSL Developer** profile is expected to have a medium to high knowledge regarding DSL development and has the highest priority to be evaluated, as it represents a primary group of users which are considered to benefit with the adoption of USE-ME conceptual framework.

The **MDD Engineer** profile is expected to have a medium to high knowledge regarding modelling techniques and has a medium priority to be evaluated, as it represents a group of users related to MDD, and USE-ME support was built using this approach.

The **Usability Engineer** profile is expected to have a medium to high knowledge regarding usability and has a low priority to be evaluated. This is because the Usability Engineers are not often included in DSL development, so they do not represent a primary group of users. When there is a possibility to introduce them into the development process of a DSL they will play a role that fits well with the tasks supported by USE-ME. It should be noted that Usability Engineers will often not be experienced, or even trained in modelling, or DSL development.

The **DSML Developer** profile is a child profile of DSL Developer and MDD Engineer. It has a high priority as it is a primary user of a USE-ME conceptual framework, having at least medium knowledge about DSL development and MDD.

8.1.2 Context environment

While developing the USE-ME framework we took into account the following environmental considerations (which are represented in Figure 8.2):

- Technical Environment - The USE-ME support is built using MDD approach and is designed to be run over a Modeling environment which requires an Operatitng System (OS). The support was built using EMF and the visual representation of models

is supported by Sirius. The USE-ME is meant to be used over any OS supporting EMF and Sirius (e.g. Windows, Mac or Linux).

- Social Environment - The USE-ME is developed and presented in the English language.
- Physical Environment - To use USE-ME a Computer should be used, having a RAM and Processor power which is required by the Modeling environment. From Interaction devices, it is mandatory to use the mouse and keyboard.

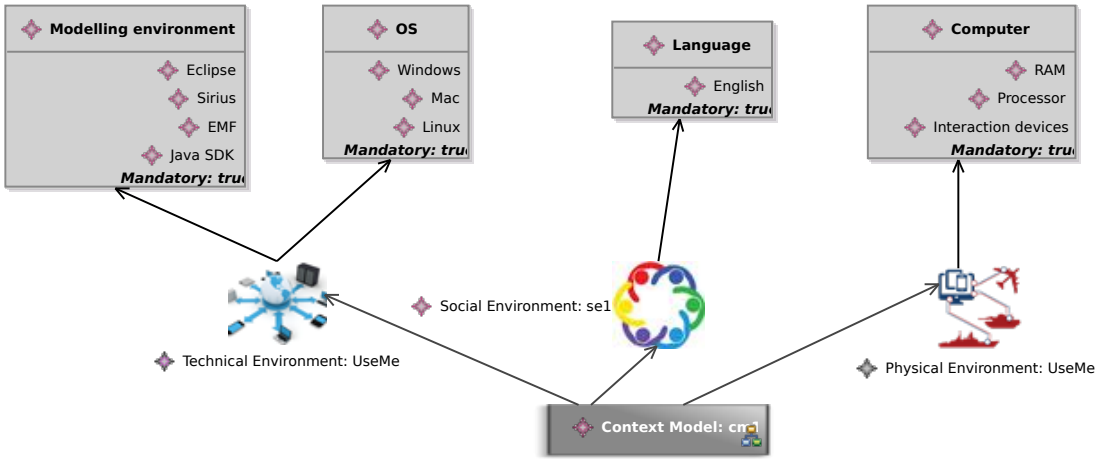


Figure 8.2: USE-ME Context Environment diagram

8.1.3 Workflows

During the development of USE-ME prototype we found the following three workflows to be mandatory:

W1 DSL `Usability Evaluation` - represent the main objective of the USE-ME framework for any end user and is prioritized as High, indicating that should be evaluated in the development cycle addressed in the context of this thesis.

W2 `Integration with` DSL `development artifacts` - the idea of the USE-ME conceptual framework is to be integrated with the DSL development cycle (see Figure 8.3). It is beneficial for any DSL developer to integrate existing development artefacts and enable an information exchange to assure the real-time updates and traceability of the impact of usability evaluation to complete DSL development scope.

W3 `Integration with experimental analysis tools` - The USE-ME conceptual framework is designed in a way to connect its testing instruments with third party applications for survey design, events capturing, or data analysis, among others. It is

expected to support an integration between specifications provided by USE-ME and and by existing experimental support (e.g. importing/exporting the questionnaire forms automatically between USE-ME and Google Forms).

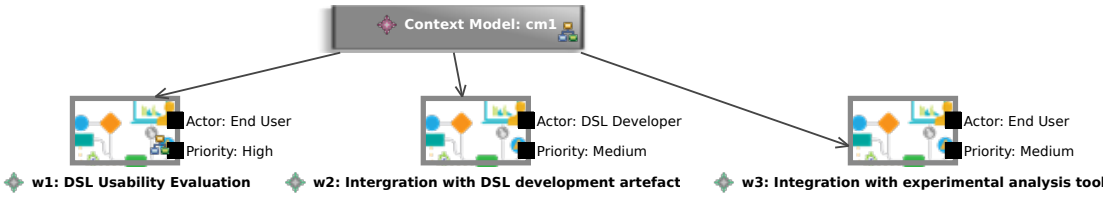


Figure 8.3: USE-ME Workflows diagram

We break the workflow W1 (see Figure 8.4) into the several independent scenarios specified for USE-ME conceptual framework in a form of activity diagrams (see Chapter 5), namely Context Modelling, Goal Modelling, Evaluation Modelling which includes Survey Modelling or/and Interaction Modelling, and Result Modelling.

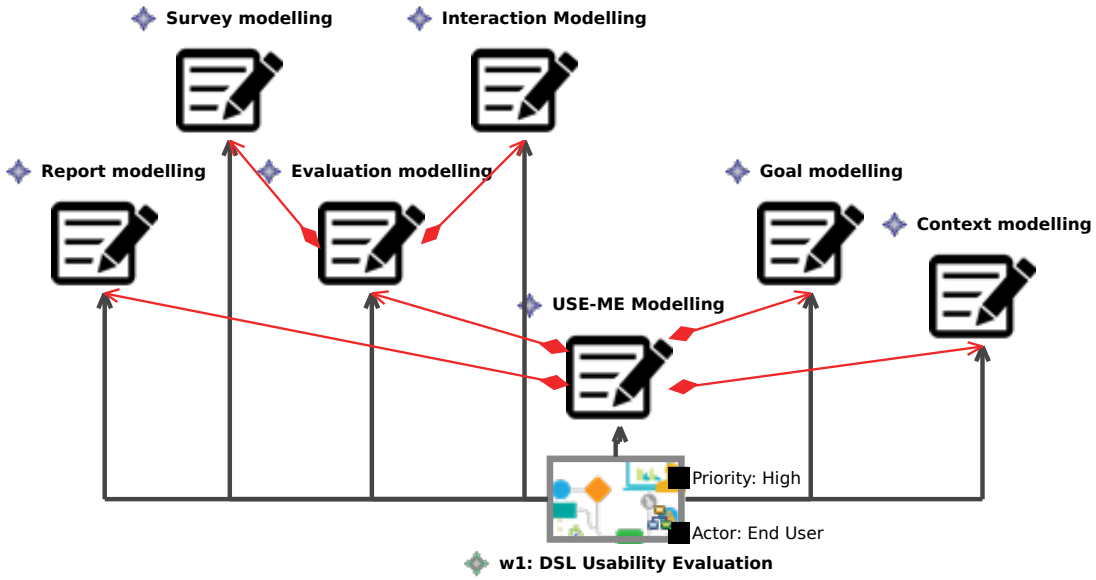


Figure 8.4: USE-ME Scenario diagram

8.1.4 Goal model

The main objective of developing the USE-ME conceptual framework was to address three research questions which were introduced in a Section 1.2. We proposed USE-ME methodological conceptual framework and associated prototype tool as solutions to our research problems. Achieving a high level of quality in use of the USE-ME conceptual framework is very important in order to be adopted by the intended community (i.e. USE-ME potential End User presented in Figure 8.1). In order to achieve a high level

of usability, the conceptual framework should be presented in a comprehensive way, as well as supported with a tool which is understandable from the perspective of End Users, especially DSL developers.

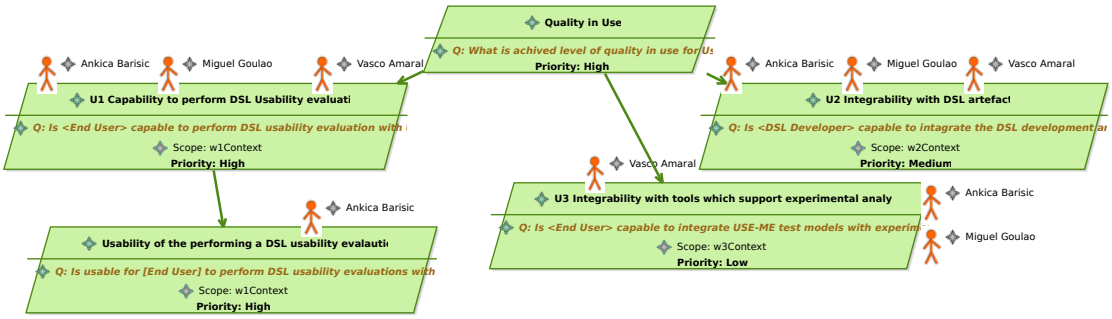


Figure 8.5: USE-ME goal model

In the context of the work presented in this thesis, we evaluate the usability goal associated with the workflow **W1** 'DSL Usability Evaluation' (Figure 8.4). All stakeholders included in USE-ME development are responsible for achievement of this goal (see Figure 8.5), however, we define its subgoal for which just Evaluation Expert is responsible for 'Usability of the performing DSL Usability evaluation'. The first step in achieving this goal was to capture all the relevant concepts and provide comprehensive specifications. This way we ensure to build the language which is *expressive* enough to support various activities which are necessary to be performed for different types of usability assessments. Further, we built supporting tool which is feasible to instantiate a usability evaluation assessment into model (Chapter 6).

The second goal is defined to address the 'Integrability with DSL artefacts', and is important to be addressed in later phases of development. By achieving this goal we expect to facilitate the process of reusing already obtained information about DSL development, which is stored within its artefacts. For instance, information which shapes the context definition can already be found in DSL artefacts like the feature diagrams, use-case/scenario descriptions, process documentation, etc. The usability goal model by itself ideally should be integrated with an existing goal model of the DSL, in which it is enabling the specification and assessment of context-aware goals.

Finally, the third goal is defined to address the 'Integrability with tools which support experimental analysis', which should enable the <End User> to automatically generate evaluation instruments, and import the results in the result models. This integration possibility will enable faster and safer implementation of the test models and save the time in importing the obtained data.

8.2 USE-ME evaluation model

To evaluate the *feasibility* of adopting USE-ME conceptual framework, we developed tool support integrable into DSL development infrastructure [27] and illustrated an instantiation of prototype models on the Visualino case study [26]. The prototype was validated in the context of projects for a DSLs graduate course [28], helping novice language engineers (i.e. master students) to prepare DSL evaluations (see Chapter 6).

To evaluate how well the USE-ME conceptual framework and associated prototype addressed our research problems, presented in Section 1.2, we gathered opinions from more experienced potential USE-ME users. We performed our evaluation with two complementary instruments:

- *Evaluation interview.* A detailed interview will be run with people who used our systematic approach in its current, or an earlier phase. We have contacted the candidates by their personal contact and shared with them the conceptual framework proposal (Annex IV). The objective of the interview was to obtain feedback about participants experience while applying usability evaluation methods in the context of the DSL they were developing, as well as, to obtain a general feedback about the USE-ME framework.

- *Evaluation survey.* The evaluation questionnaire will be run with people which are experienced in MDD and DSL development or/and with HCI and empirical studies.

8.2.1 Evaluation subjects and context

The evaluation targets people experienced in research, who obtained the master or doctoral degree. The participants are expected to fit some of the [End User] profiles, which have a relevant knowledge in usability, modelling or DSL development (see Section 8.1.1).

As we targeted more senior people in research, who are usually not available to spend a lot of time, we decided to run the evaluation survey [Online] and keep it short as possible. It was not necessary to set up a technical environment, although the tool was used in a presentation. However, participants were expected to speak English and needed to use a computer and a web browser and web access.

We invited people with the following profiles to participate:

- people familiar with our systematic approach in its current or an earlier phase (e.g. ones that were making part of DSL development for the case studies where we applied USE-ME conceptual framework, or were applying it independently in its early phase);

- people from the networks related to DSL development like DSM-TP[1] summer school, MPM4CPS[2] COST IC1404 action members or the DSM Forum[3];
- people we cited in our references for which we had e-mail, or a profile in social networks like LinkedIn [4] and Research Gate[5];
- people recommended by the other participants.

In total we invited over 350 persons using their personal contact reference. Also, we made the link accessible at USE-ME GitHub wiki page [6] and all participants were invited to share the link further. We reproduce here the letter of invitation which was shared with potential participants.

'Invitation to provide a feedback about usability evaluation modelling framework for Domain-Specific Languages'

'SURVEY LINK: https://goo.gl/forms/Js4Nh8V6VCZvmsAB2

This experimental work is conducted within the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) in the context of the evaluation of USE-ME conceptual framework which was developed as a part of the PhD thesis.

We hope that you may find it interesting to contribute by providing your opinion about the provided solution. We are searching mainly for people which have background knowledge about domain-specific languages, model-driven development, human-computer interaction, empirical studies OR usability testing. However, anyone interested in the topic is welcome to participate. I would be also grateful if you could share the invitation link also with colleagues, for which you believe that may find this work interesting.

The experiment will be kept strictly confidential and will be made available only to members of the research team of the study or, in case external quality assessment takes place, to assessors under the same confidentiality conditions. Data collected in this experiment may be part of the final research report, but under no circumstances will your name or any identifying characteristic be included in the report. In particular, there is no intention of judging you as a person or the skills and experience that you will use in this survey - the goal is the evaluation of the proposed approach!'

8.2.2 Evaluation objectives

The objective of this study was to evaluate the goal U1 'Capability to perform DSL usability evaluation' in regard to If it is *feasible* for <DSL Developers> to perform DSL usability evaluation with USE-ME. We associate with this goal our three research questions defined

[1] msdl.cs.mcgill.ca/conferences/dsm-tp-2017 (accessed September 19, 2017)
[2] http://mpm4cps.eu/ (accessed September 19, 2017)
[3] http://www.dsmforum.org/ (accessed September 19, 2017)
[4] www.linkedin.com (accessed September 19, 2017)
[5] www.researchgate.net (accessed September 19, 2017)
[6] github.com/akki55/useme/wiki (accessed September 19, 2017)

in (Section 1.3) and define the following evaluation goals and their associated question as follows:

G1 Does the USE-ME conceptual framework enable End Users to model DSL usability evaluations?

G2 Does the USE-ME conceptual framework enable End Users in promoting usability concerns since an early stage of DSL development

G3 Does the USE-ME tool support enable End Users to built usability evaluation into the development process of the DSL

The evaluation goal [**G1**] is associated with RQ1, and we specify the problem which it aims to solve in form of Goal-Quality-Metric (GQM) [198] format: *'Analyze the effect of the [USE-ME conceptual framework], for the purpose of evaluation, with respect to its impact on the [feasibility] to model the DSL usability evaluation, from the point of view of the [researcher], in the context of the evaluation survey conducted with [End User].*

The evaluation goal [**G2**] is associated with RQ2, and associated problem statement is defined as: *'Analyze the effect of the [USE-ME conceptual framework], for the purpose of evaluation, with respect to its impact on [feasibility] to promote usability concerns since an early stage of DSL development, from the point of view of the [researcher], in the context of the online evaluation survey conducted with [End User].*

The evaluation goal [**G3**] is associated with RQ3, and we specify problem which it aims to solve in form of Goal-Quality-Metric (GQM) as: *'Analyze the effect of the [USE-ME tool support], for the purpose of evaluation, with respect to its impact on [feasibility] to built usability evaluation into the development process of the DSL, from the point of view of the [researcher], in the context of the online evaluation survey conducted with [End User].*

8.2.3 Evaluation process and documentation

In Figure 8.6 we introduce the process and documents which were used during the survey execution. All participants needed to watch the presentation video[7] about the USE-ME conceptual framework. We prepared a 15 minutes video in which we introduced the motivation for developing the USE-ME conceptual framework, its usage which was illustrated by a specification model from Chapter 5, and an instantiation of the evaluation model supported by the USE-ME prototype from Chapter 6. All participants were asked to respond the survey questionnaire which we introduce in Section 8.3. Finally, they were provided with the USE-ME tool [27] and the article describing USE-ME conceptual framework [26] to consult additionally if they would like to invest more time to understand the conceptual framework.

The pilot trial of the survey was executed with our interview candidates and three members of NOVA-LINCS research centre. It was run in the period from July 1, 2017

[7] youtu.be/RjIGFex-zQM (accessed September 19, 2017)

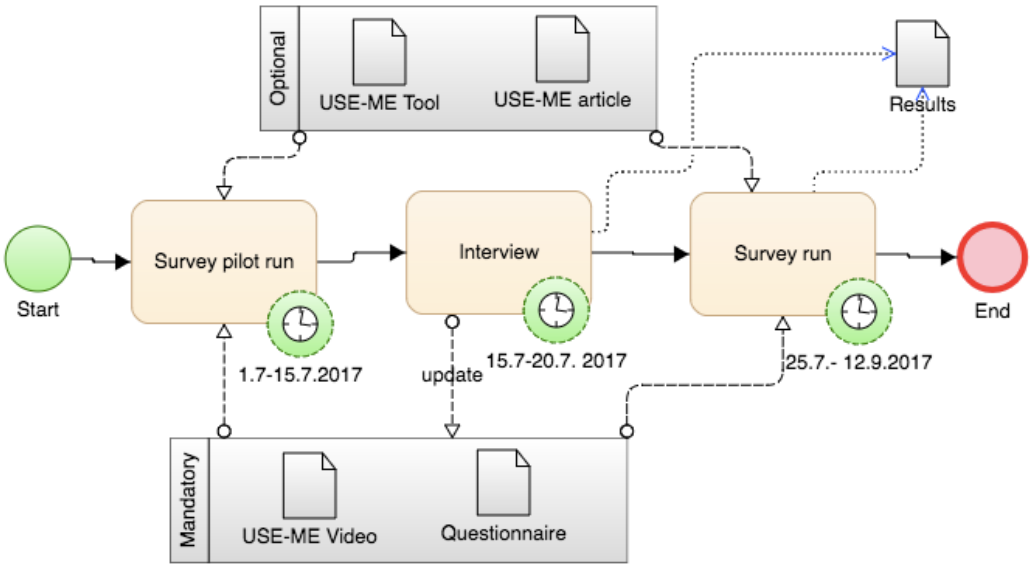


Figure 8.6: Evaluation process

till July 15, 2017. The participants were asked to access the provided documentation, watch the video tutorial and to answer the survey. All participants from the pilot run rated the materials as complete and easy to understand, and provided small improvement suggestions. Evaluation materials were updated in accordance with provided suggestions after the pilot session was over.

The interview with participants was scheduled in the period from July 15 until July 20, 2017. Interview sessions were conducted individually and participants were asked to allow to be voice recorded. This helped the evaluator to analyse the feedback afterwards and by that shorten the time and increase the quality of the interview. The interpretations which were taken from the interview were given to participants for review to confirm that they were well understood by the evaluator.

Finally, we run a survey from July 25 till September 12, 2017. after which we analysed and reported the results (see Section 8.6).

8.3 Survey model

The survey form is reproduced in Annex VII. The survey questions are classified in Table 8.1 and categorized as background and feedback questions. Background questions are designed to collect the participant's demographics (DQ) and self-rated experience in the area (EQ). Feedback questions (FQ) are meant to assess the opinion about the motivation and feasibility of the USE-ME conceptual framework. Participants were invited first to read an abstract, watch the video tutorial and to check the documentation regarding the USE-ME conceptual framework. Like this, they were able to decide upfront if they fit the right profile to provide feedback. After this, participants are asked to fill in the

Table 8.1: Question classification

Classification:	Questions:	Classification:	Questions:
BACKGROUND	**DQ,EQ**	**FEEDBACK**	**FQ**
Demographics	DQ1-DQ9	**Motivation**	FQ4-FQ8
Experience	EQ1-EQ23	**Feasibility**	FQ9-FQ18
Modelling	EQ4-EQ5	RQ1	FQ9-FQ12
UML	EQ6-EQ10	RQ2	FQ13-FQ14
DSL	EQ11-EQ15	RQ3	FQ15-FQ18
Requirement engineering	EQ16-EQ17	**Suggestions**	FQ19
Agile development	EQ18	**Usability tools**	FQ20
HCI	EQ19-EQ23		

Table 8.2: Scales definition

DegreeScale:	ExpertScale:	AgreeScale:	DevelopmentScale:
PhD	Expert	Strongly agree	Yes, I developed few functional DSLs of which at least one is widely used
MSc	Advanced	Agree	Yes, I developed a functional DSL
BSc	Intermediate	Don't know	Yes, I developed few prototypes
Other	Emerging	Disagree	Yes, in the context of faculty course
	None	Strongly disagree	No

Background questionnaire, followed by the Feedback questionnaire. Finally, they were asked to report on the time which they spent into getting familiar with an conceptual framework, as well the time to fill in the questionnaire.

In order to quantify answers to our questionnaire, we predefined four different Likert scales in Table 8.2. We used DegreeScale, development scale and experts call as a part of Background Questionnaire. The degree scale was meant to obtain the feedback about the obtained academic degree of our participants. The ExpertScale served for reporting a level of their experience with the relevant approaches or tools. The development scale was used for participants to report their experience with the DSL development. We preferred to define this scale in a manner which will reflect the participant's practical experience with the DSL development. Finally, we used AgreeScale in the Feedback Questionnaire.

8.3.1 Background questionnaire

The background questionnaire, defined in the Table 8.3, served for collecting demographics and experience data from the survey participants. The questions are designed to assess the [End User] profile characteristic, which we defined for USE-ME context model in Section 8.1.1. Information obtained by DQ2, DQ3, DQ4 and DQ9 are kept strictly confidential and served for validation of the persons profile and as a personal contact point. Questions DQ5, DQ6, DQ7 and DQ8 served to highlight the diversity of our participants, as well as their maturity level.

Table 8.3: Background Questionnaire

Id	Question	Value
DQ1	Timestamp:	Number
DQ2	Email Address:	Text
DQ3	Full Name:	Text
DQ4	Personal page or public profile:	Text
DQ5	Age:	Number
DQ6	Country:	Text
DQ7	Degree:	DegreeScale
DQ8	Current Work/Research Position::	Text
DQ9	Current Work/Research Institution:	Text
EQ1	Experience background:	Academic / Industry
EQ2	How many years of working experience do you have?	Number
EQ3	How many years of research experience do you have?	Number
EQ4	How would you rate your level of knowledge related to modelling techniques?	ExpertScale
EQ5	How would you rate your experience related to Model-Driven Development MDD)?	ExpertScale
EQ6	How would you rate your level of knowledge regarding UML?	ExpertScale
EQ7	How experienced are you with modelling use cases?	ExpertScale
EQ8	How experienced are you with UML activity or process diagrams?	ExpertScale
EQ9	How experienced are you with UML class diagrams?	ExpertScale
EQ10	Are you familiar with modelling interaction (communication) diagrams?	ExpertScale
EQ11	How would you rate your level of knowledge related to Domain-Specific Languages(DSL)?	ExpertScale
EQ12	Did you ever develop a DSL?	DevelopmentScale
EQ13	How would you rate your level of knowledge regarding the Eclipse working environment?	ExpertScale
EQ14	How experienced are you with the Eclipse Modeling Framework (EMF)?	ExpertScale
EQ15	How experienced are you with the Sirius Modeling tool?	ExpertScale
EQ16	How experienced are you with software requirements engineering?	ExpertScale
EQ17	How familiar are you with goal-oriented requirements approaches?	ExpertScale
EQ18	How experienced are you with agile development?	ExpertScale
EQ19	How would you rate your level of knowledge regarding Human-Computer Interaction?	ExpertScale
EQ20	How experienced are you with User-Centered design techniques?	ExpertScale
EQ21	How familiar are you with empirical experiments?	ExpertScale
EQ22	How familiar are you with usability testing?	ExpertScale
EQ23	Can you describe (refer to) your previous HCI experience?	Text

Experience questions reflect a self-rated Expertise scale (*ExpertScale*) of the participant regarding his/her knowledge of skills which are relevant for understanding and applying the USE-ME conceptual framework. These skills are classified into a knowledge of Modelling, UML, DSL, Requirement Engineering, Agile development and HCI, as we classified in Table 8.1. The combination of this skills reflects the experience sets which are expected from the language engineer or/and the expert evaluator, participating in any DSL development project, according to USE-ME.

First, we asked the participants to report on their working and research experience (EQ1, EQ3). To assess the participant's knowledge about Modelling we ask them to rate their experience of modelling in general, as well as with the MDD (EQ4, EQ5). Further, we asked participants to report on their knowledge regarding the UML in general (EQ6) and experience with different UML diagrams (EQ7, EQ8, EQ9, EQ10). The next question aimed to assess the experience of participants with DSL in general (EQ11), as well as with practical development of DSLs (EQ12), in particular, knowledge of Eclipse working environment, EMF and Sirius (EQ13, EQ14, EQ15), which were the tools which used while developing the USE-ME prototype. Further, we assessed the participant's knowledge regarding requirement engineering (EQ16, EQ17) and agile development (EQ18). Finally, we wanted to assess their background regarding HCI (EQ19), of which in particular the UCD, empirical experiments and usability testing (EQ20, EQ21, EQ22).

8.3.2 Feedback questionnaire

Feedback questions, defined in Table 8.4, are designed to collect motivation and opinions of the participants regarding the USE-ME conceptual framework, in most of the cases by using an Agreement Scale (*AgreeScale*). Motivation questions reflect how participants perceive the importance of the presented problem. First, we assess with which of the provided materials the participants got familiar with (FQ1, FQ2, FQ3), for which it was mandatory to watch the provided presentation video.

Further, we asked participants to provide a feedback about the need for a conceptual framework such as ours (FQ4, FQ5, FQ6), as well as the need for a context-dependent and reusable approach (FQ7, FQ8). Next, we wanted to obtain feedback about how the USE-ME conceptual framework impacts our evaluation objectives defined in a Section 8.2.2, which answers to research questions of this thesis. The question FQ9 assesses the perceived level of the USE-ME conceptual framework to support the DSL usability evaluations, as well as the relevance of provided concepts (FQ10), if the conceptual framework is easy to understand (F11) and if it is general enough to be applied to different DSL development practices (F12). Next, we obtained feedback concerning whether if the conceptual framework is suitable to be applied from early phases of DSL development (F13), and if DSL development can benefit from the iterative application of conceptual framework for large users groups (F14), which should address our RQ2. Further, the questions F16, F17 and F18 obtained feedback about the feasibility, expressiveness and integrability of the

Table 8.4: Feedback Questionnaire

Id	Question	Value
FQ1	Did you watch the presentation video?	Yes/No
FQ2	Did you read the paper presenting the conceptual framework?	Yes/No
FQ3	Did you try to use the tool?	Yes/No
FQ4	There is a lack of systematic approach for evaluation of DSLs	AgreeScale
FQ5	Usability evaluations are necessary for a DSL development in practice	AgreeScale
FQ6	Current usability evaluations of DSLs are too expensive and not reusable.	AgreeScale
FQ7	It is necessary to specify explicitly a context of the DSL when evaluating its usability	AgreeScale
FQ8	DSL evolution cycle should include usability re-evaluation	AgreeScale
FQ9	The provided approach supports modelling of the usability evaluation process for DSLs	AgreeScale
FQ10	Concepts modelled by the USE-ME framework are relevant for DSL development	AgreeScale
FQ11	The USE-ME approach is easy to understand	AgreeScale
FQ12	The approach is independent of the particular DSL development approach	AgreeScale
FQ13	Approach is suitable to be applied from the early stage of the development of the DSL	AgreeScale
FQ14	DSLs targeting large user groups can benefit from the investment in application of the USE-ME approach iteratively	AgreeScale
FQ15	The USE-ME tool makes it feasible for a DSL engineer to model a usability evaluation	AgreeScale
FQ16	The USE-ME tool is expressive enough for specifying usability evaluation of DSL	AgreeScale
FQ17	The USE-ME tool supports the integration of usability evaluation approach into development process of the DSL	AgreeScale
FQ18	Investment into the development of the USE-ME prototype tool into real product is worthy	AgreeScale
FQ19	Can you please provide your suggestions concerning how to improve the USE-ME tool or approach itself:	Text
FQ20	Are you familiar with any other tool which supports usability evaluation?	Yes/No
FQ20_1	What are the other tools you are familiar with?	Text
FQ20_2	The USE-ME is more suitable then alternatives for usability evaluation of DSLs	AgreeScale
FQ20_3	The USE-ME is more complete then alternatives for usability evaluation of DSLs	AgreeScale
FQ21	How much time did you spend on getting familiar with approach (watching video, reading the article and/or trying the tool)? (in minutes)	Number
FQ21	How much time did you spend on answering this questionnaire? (in minutes)	Number

USE-ME support. Finally, we asked participants to report if they were familiar with other tools which support usability evaluations.

8.4 Background analysis

The survey was run in period of July 1 2017 until September 15 2017 (see Figure 8.6). We obtained answers from 53 participants, from which 8 were participating in a pilot run. We integrate the pilot results, as no significant changes were performed in the survey materials that would influence the answers. Only things that was changed after survey run was addition of optional clarification questions in a case when participant disagree with our statements from feedback questionnaire (FQ). However, we needed to eliminate the answers from our analysis related to one of the participants who did not report that he/she watched the presentation video. Therefore, in total, we report on feedback obtained from 52 participants.

8.4.1 Demographics

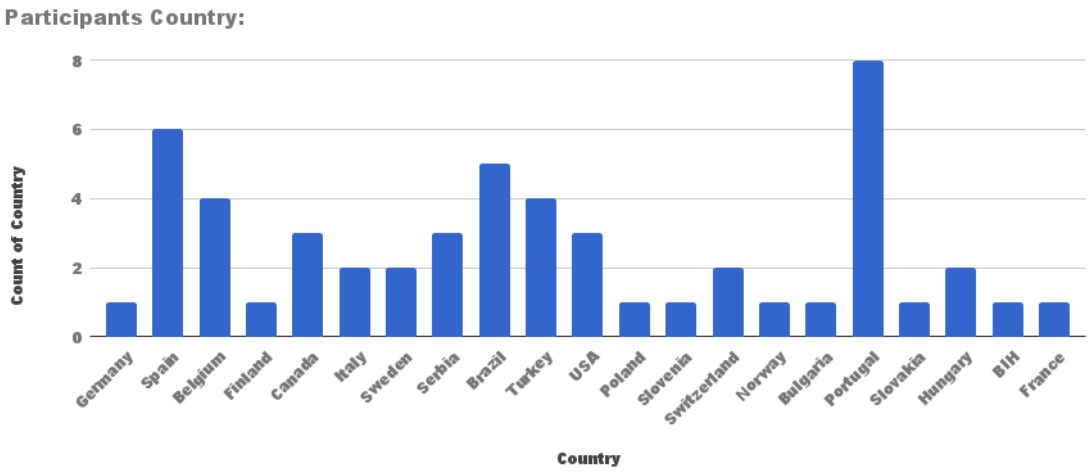


Figure 8.7: Participants Country (DQ6)

The participants had the age range from 24 to 69 years, and they are coming from 21 different countries (see Figure8.7), from which the most represented were Portugal, Spain, Brazil, Belgium and Turkey.

Most of the participants have a PhD degree (57.7%) while rest have an MSc degree (see Figure 8.8). In the Figure 8.9 we can see that 38% of participants are currently PhD candidates, 26% are university professors, 22% were PostDoc researchers while 14% reported to be working in industry. We should note that some of the participants were working in both academy and industry, and we presented the graph with their academic oriented positions when they were reported.

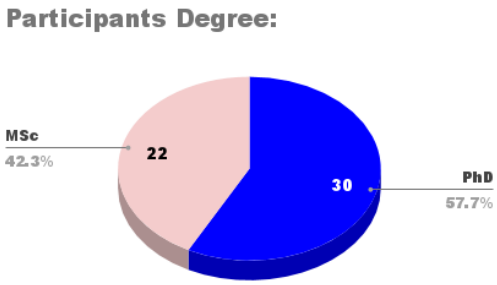


Figure 8.8: Participants Degree (DQ7)

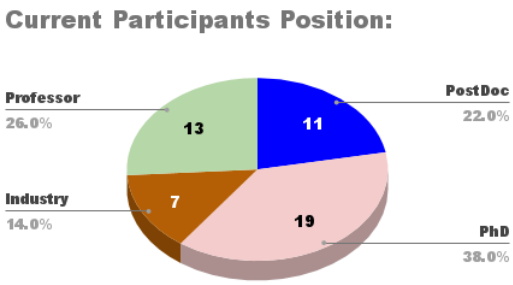


Figure 8.9: Participants current position (DQ8)

8.4.2 Experience

While all of the participants reported to had an academic experience background (EQ1), 26 (49%) also reported having industry experience. In Figure 8.10 we present participants work/research experience in years. We obtained results from 17.3% of participants which are not having a lot of research/work experience (0-5 years), 28.8% were having intermediate experience (6-10 years). Most of the participants (30.8%) are advanced, having 11-20 years of experience, while 23% are senior; six having 21-30 years of experience, while other six reported having more than 30 years of working/research experience.

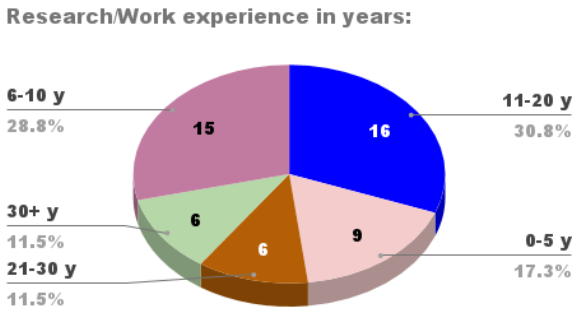


Figure 8.10: Participants Working/Research experience (EQ2/3)

8.4.2.1 Modelling

Most participants reported to be Expert (34.6%) or Advanced (48.1%) concerning their knowledge related to modelling techniques (see Figure 8.11), 11.5% were Intermediate while only 5.8% were Emerging. This indicates there were no participants which were not having modelling experience, meaning that they should be familiar with the modelling approach which was used for the USE-ME development.

Regarding the reported experience with MDD, we had 28.8% Expert participants, 42.3% Advanced, 19.2% Intermediate, 9.6% Emerging and there were no participants without an experience in MDD.

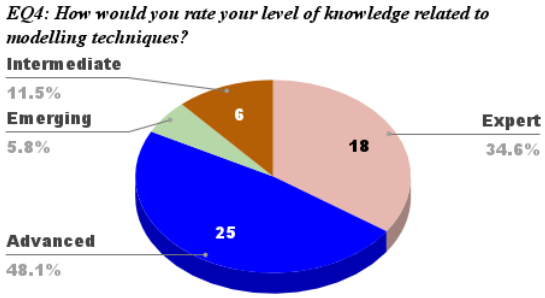


Figure 8.11: Modelling experience (EQ4)

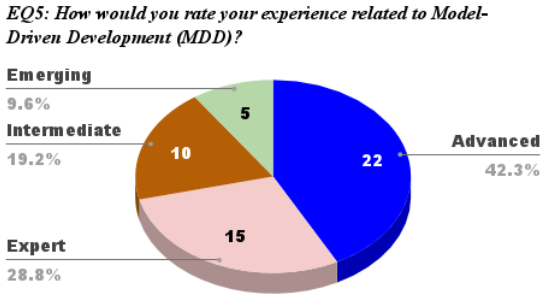


Figure 8.12: MDD experience (EQ5)

8.4.2.2 UML

In Figure 8.13 we can see that all participants except one had knowledge regarding UML. The 19.2% participants reported being Experts, 48.1% Advanced, 26.9% Intermediate and one as Emerging. This indicates, that participant should not have problems in understanding the specification diagrams of USE-ME, which are defined using a UML syntax.

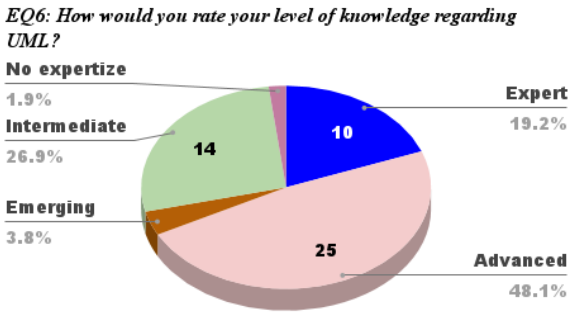


Figure 8.13: UML experience (EQ6)

All participants with UML experience were familiar with use case modelling (Figure 8.14), activity/process diagrams (Figure 8.15) and class diagrams (Figure 8.16). Around 70% reported to be Advanced or Experts regarding their experience with this UML diagrams. However, for interaction diagrams (Figure 8.17), three additional participants reported to have no expertise, but still 50% were Advanced or Experts.

8.4.2.3 DSL

Further, 30.8% of participants rated their level of knowledge regarding DSL as Expert, 38.5% to be advanced, 25% Intermediate, 3.8% Emerging and one reported not to have experience (Figure 8.18). However, the participant with no experience with a DSLs reported to have a relevant knowledge about usability and modelling techniques, and considering his/hers senior status (professor position and 28 years of work/research experience) we took his/her feedback as relevant for this study.

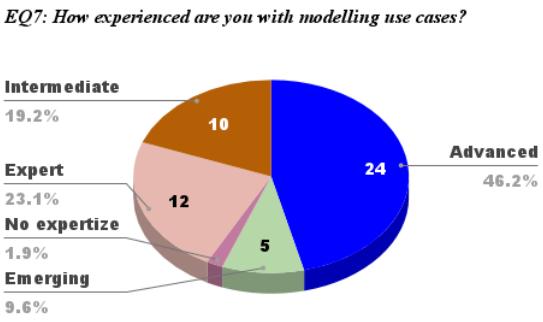


Figure 8.14: Use cases experience (EQ7)

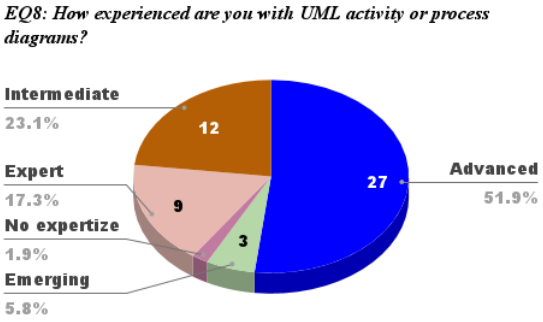


Figure 8.15: Activity/process diagrams experience (EQ8)

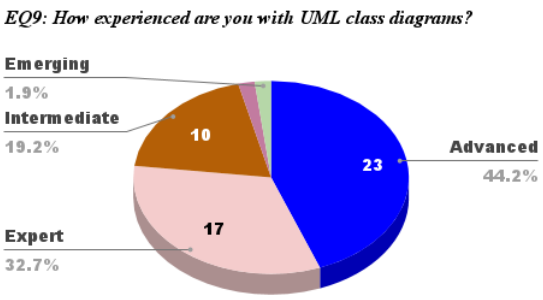


Figure 8.16: Class diagrams experience (EQ9)

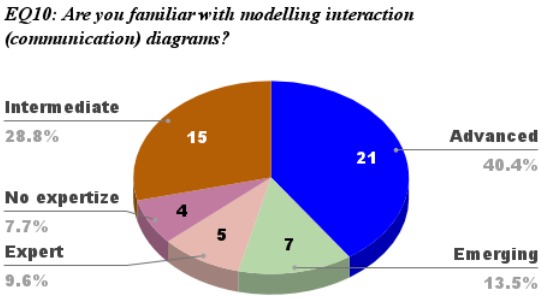


Figure 8.17: Interaction diagrams experience (EQ10)

Considering DSL development experience, 23.1% of participants reported that they developed functional DSLs of which at least one is widely used, indicating they were Experts. 25% reported they developed a functional DSL, marked as Advanced, 38.5% developed few prototypes, indicating Intermediate level, 3.8% developed DSL in the context of faculty course, while 9.6% of participants didn't have practical experience in DSL development (Figure 8.19).

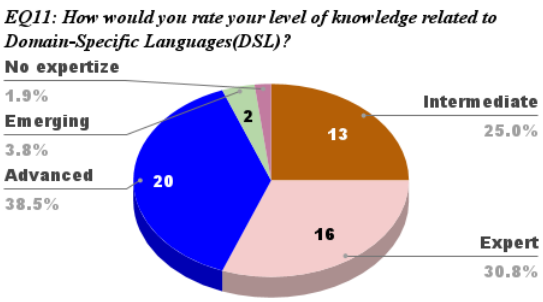


Figure 8.18: DSL experience (EQ11)

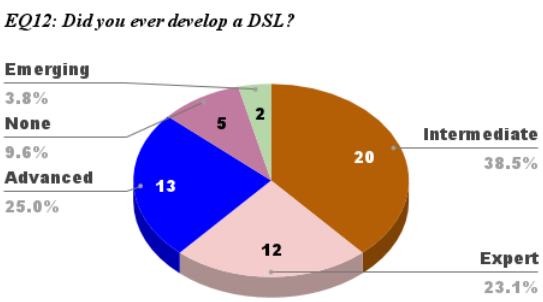


Figure 8.19: DSL development level (EQ12)

As the USE-ME prototype was developed with EMF and Sirius, we report on the

participants experience relevant to this technology which makes a part of the technical context for USE-ME support. Although it was not mandatory to test the tool support, the understanding of USE-ME diagrams and restrictions in providing certain futures in relation to the used technology, is expected to be deeper to those who had more experience with this tools.

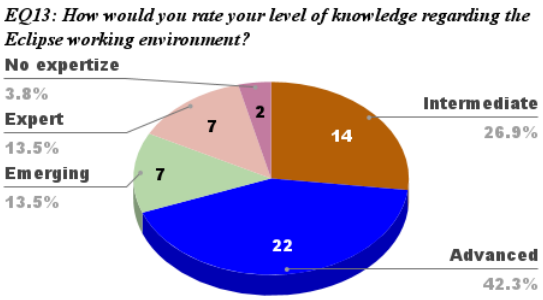


Figure 8.20: Eclipse experience (EQ13)

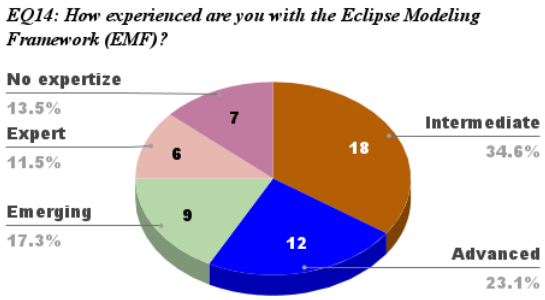


Figure 8.21: EMF experience (EQ14)

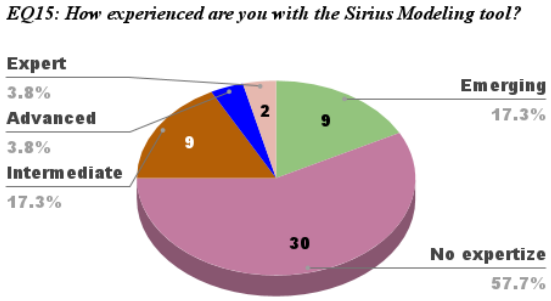


Figure 8.22: Sirius experience

For background knowledge regarding the underlying platform, Eclipse, participants rated their knowledge as follows (Figure 8.20): 13.5% are Experts, 42.3% Advanced, 26.9% Intermediate, 13.5% Emerging and 3.8% had No Expertise. Concerning the knowledge of the EMF, 11.8% of participants report to be Experts, 23.1% Advanced, 34.5% Intermediate, 17.3% as Emerging and 13.8% having no expertise (Figure 8.21). Finally, the 3.8% of participants are Experts with Sirius, 3.8% Advanced, 17.3% Intermediate, 17.3% Emerging while 57.7% had no experience with a Sirius tool (Figure 8.22).

8.4.2.4 Requirements engineering

In this section, we report on participants background regarding requirements engineering, which we find important as the usability is seen as a non-functional requirement in the requirement engineering.

In Figure 8.23 we can see that 15.4% participants rated them as Experts regarding their requirements engineering knowledge, 38.5% Advanced, 32.7 Intermediate, 11.5%

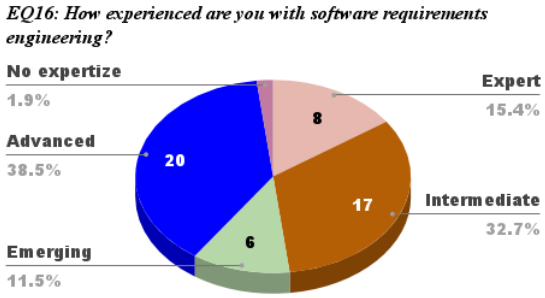


Figure 8.23: Requirements engineering experience (E16)

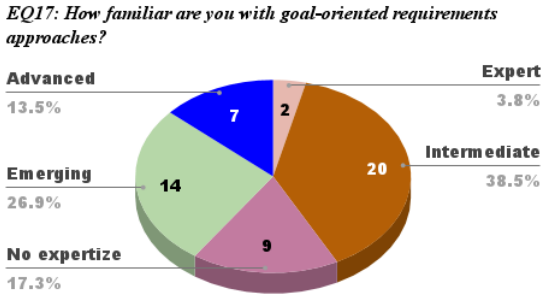


Figure 8.24: Goal modelling experience (E17)

Emerging while just one had no expertise. Concerning the goal-oriented approaches (Figure 8.24), 3.8% reported to be Experts, 13.8% Advanced, 38.5% Intermediate, 26.9% Emerging, while 17.3 % had no expertise. Knowledge of goal modelling is expected to give higher credibility to participants when reasoning about usability goal model, which is a central part of USE-ME framework.

8.4.2.5 Agile development

The knowledge about agile development is expected to enable participants to envision the iterative incremental development process which is proposed when applying a USE-ME conceptual framework.

In Figure 8.25 we can see that only one participant didn't have any experience with agile and only 9.6% were emerging. The rest of the participants rated their knowledge as relevant, namely 5.8% Expert, 30.8% Advanced and 51.9% Intermediate.

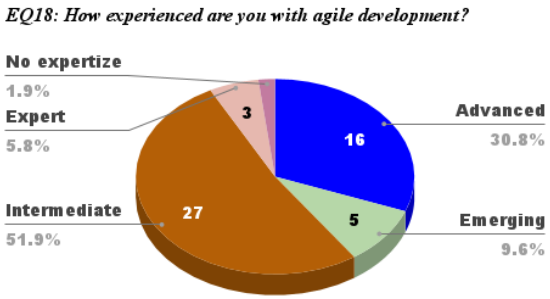


Figure 8.25: Agile development experience (E18)

8.4.2.6 HCI

Finally, we report about HCI experience from our participants, which indicates their level of expertise with usability. We can see in Figure 8.26 that 5.8% of participants self rated

themselves as Expert, 19.2% as Advanced, 40.4% as Intermediate, 25% as Emerging and only 9.6% of participants had no expertise in HCI.

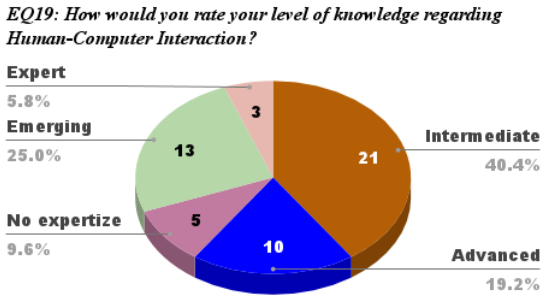


Figure 8.26: HCI experience (EQ19)

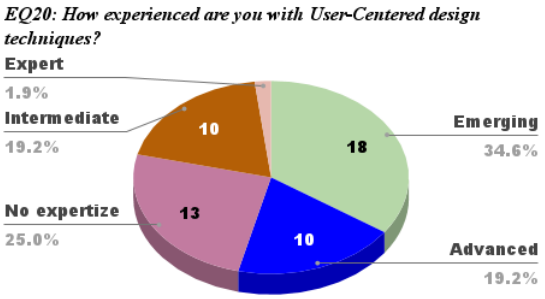


Figure 8.27: UCD experience (EQ20)

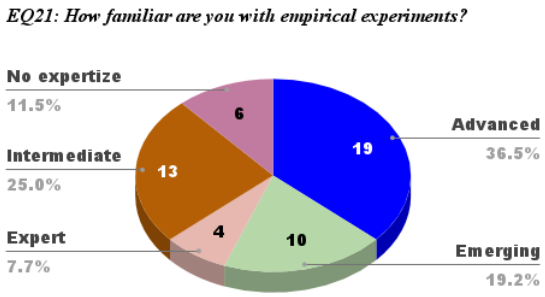


Figure 8.28: Empirical experiments experience (EQ21)

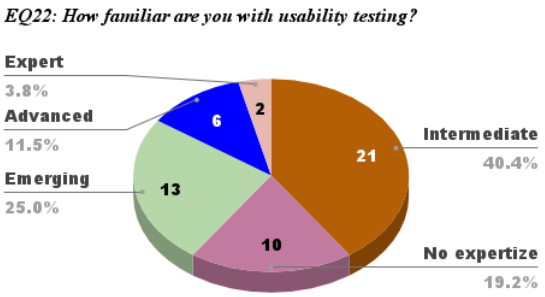


Figure 8.29: Usability testing experience (EQ22)

In Figure 8.27 we can see that we had just one reported Expert on UCD, while more than half participants were Emerging or Advanced. However, the 25% of participants reported not to have any expertise with UCD. For the empirical experiments, we obtained slightly better results (Figure 8.28), where we had four participants reporting as Experts, while just six of them had no expertise. Finally, we had two Experts with usability testing, 11.5% Advanced and 25.6% Emerging, while more than half were Intermediate or without expertise.

8.5 Feedback analysis

8.5.1 Motivation

In this section we report on participants feedback regarding our motivation statements which justify the importance into the investment of development of USE-ME conceptual framework. We can see in Figure 8.30 that almost all participants Agree with a statement that there is a lack of systematic approach for usability evaluation of DSLs. Only 9.6% participants reported being indifferent.

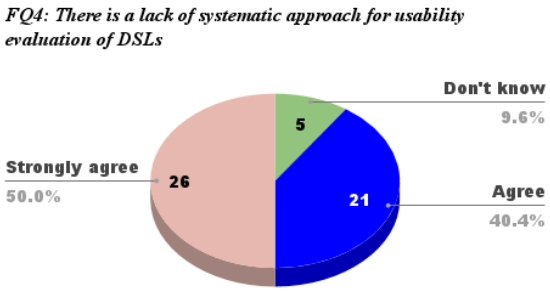


Figure 8.30: Motivation feedback (FQ4)

Most participants (96.2%) reported finding that usability evaluations are necessary for a DSL development in practice (Figure 8.31). One participant reported being indifferent, while one reported to Disagree with a statement. S/he answered to question 'Can you please let us know why do you find that usability evaluations are not necessary for a DSL development in practice?': *'Never said they are not necessary. I just disagree they are necessary in every case. In some cases, to be widely adopted, the language needs to be adapted to the user. On other cases, the user needs to adapt to the language (e.g., making a language efficient to compile for embedded devices).'* Participant pointed that evaluations are still necessary, but not in every case, with witch we agree and discussed applicability of USE-ME conceptual framework in Chapter 7.

Further, 46.2% participants reported to Agree that the current usability evaluations of DSLs are too expensive and not reusable (Figure 8.32). However, 51.9% reported not to know whether or not they are too expensive and the extent to which they are reusable. One participant reported to Disagree with a statement. S/he answered to question 'Can you please let us know why do you disagree with a statement: "Usability evaluations of DSLs are too expensive and not reusable":?' as *'Typical approach we follow is carrying out a pilot in which a number of topics on DSL are evaluated - including usability too. This means that few users will apply the DSL for typical tasks and their feedback is collected and included in the next version. This process can continue also when in production use - albeit companies don't usually do that - perhaps because the usability topics are not so big (or important for them) that they call for evaluation.'*

Most of the participants, 94.2%, reported to Agree with the statement that it is necessary to specify explicitly the context of the DSL when evaluating its usability, while two participants reported being indifferent (Figure 8.33).

Finally, 94.2%, reported to Agree that DSL evaluation cycle should include usability re-evaluations, while two participants reported being indifferent (Figure 8.34).

8.5.2 Research question 1

In this section we report on the statements related to RQ1 'How are we able to model the DSL usability evaluation?' of this thesis. We state that **We are able to model the DSL**

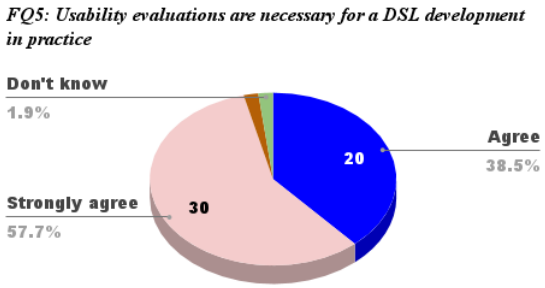


Figure 8.31: Motivation feedback (FQ5)

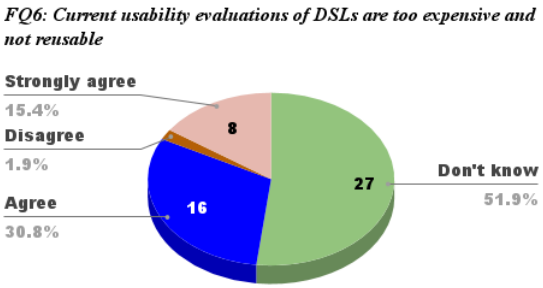


Figure 8.32: Motivation feedback (FQ6)

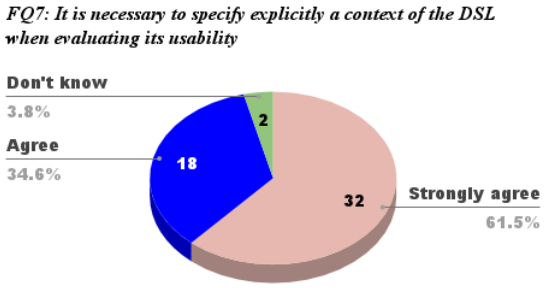


Figure 8.33: Motivation feedback (FQ7)

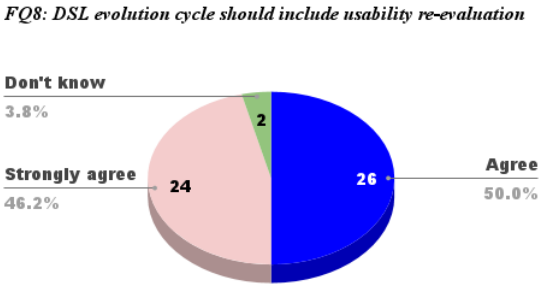


Figure 8.34: Motivation feedback (FQ8)

usability evaluations by applying a USE-ME conceptual framework.

In Figure 8.35, we can see that 92.3% of participants Agree that the USE-ME conceptual framework supports modelling of the usability evaluation process for DSLs, while the rest did not express any strong opinion. Also, 96.2% of participants Agree that the concepts modelled by the USE-ME framework are relevant for DSL development, while only two participants stay indifferent (Figure 8.36).

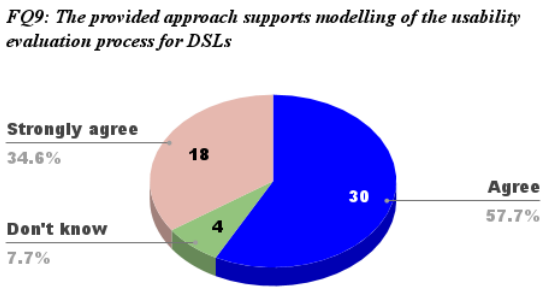


Figure 8.35: RQ1 feedback (FQ9)

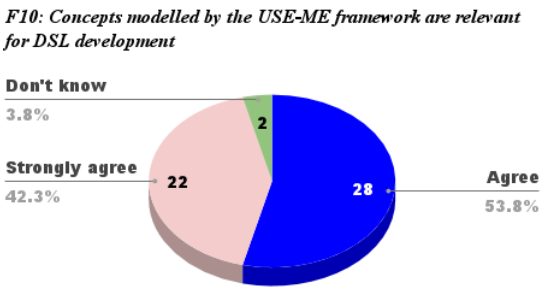


Figure 8.36: RQ1 feedback (FQ10)

Further, in Figure 8.38 we can see that 96.2% participants reported to Agree that the USE-ME conceptual framework is independent of the particular DSL development approach, while two participants stayed indifferent.

Finally, in Figure 8.37 we can see that 53.8% participants reported to Agree that

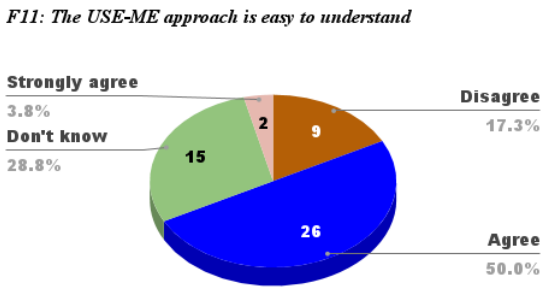


Figure 8.37: RQ1 feedback (FQ11)

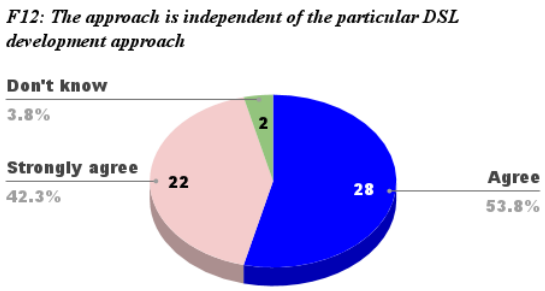


Figure 8.38: RQ1 feedback (FQ12)

the USE-ME conceptual framework is easy to understand. However, 28.8% participants reported being indifferent, while nine of them disagree. We asked participants which disagree to explain what did they find difficult, or hard to understand about the USE-ME conceptual framework. One of participants reported *'There are so many concepts and artefacts generated in each phase that turn the process intricate and very detailed. Some artefacts are created during the process, others associated with existing artefacts that make users confused during the first contact with the methodology. It requires elaboration, take notes aside, making conceptual associations to try to solve the entire puzzle. I think that after some tutorials and training material will be possible to manage the content in an easy way. However, current material is not didactic enough for a fast-paced learning.'* indicating that teaching materials should be improved in order to users learn approach in an easy way. We agree with this statement and indicate that it is necessary to evolve tool and it's documentation to make it adoptable by general audience, leading to an industrial product. Another participant noted that *'Based on the video there are many steps to be followed and if a team is defining a DSL they already have several other topics to be considered too. So from industry point of view it should be very easy to use so that practitioners would follow it'*. We find that adoption of the approach require person in the team which will be dedicated to evaluation of the product, however USE-ME is expected to lead a person in performing evaluation even if s/he lacks evaluation experience. Finally, we highlight also the following comment: *'As an expert in usability rather than DSL development I find the approach strongly oriented MDD and DSL development and modelling, which makes the approach less understandable for me. Though the video was helpful, I would expect more focus on practical usability evaluation a clear example of a scenario with users, maybe a support for filling the questionnaires by the users during a usability test. I find the user interface of USE-ME complex, but it might be my low experience with using Eclipse DSL development tools. In my research, I am focused more on practical DSL development (modelling in code), which in my opinion is faster than UML modelling. Therefore I find the approach unnecessary complex. But I think that for people-oriented on UML modelling, the tool can be useful.'*

In summary, the major suggestions to the better understandability of the approach is the improvement of the tool support for hiding, where possible, the complexity of the

USE-ME conceptual framework. However, we must note that the objective of this thesis was not providing a commercial/usable tool support, but a conceptual framework which has potential to be adopted by the community.

That said, we obviously are concerned with the usability of the USE-ME conceptual framework. An evolution of the USE-ME tool support is currently being developed in the context of an MSc dissertation [173], with an emphasis on guiding language engineers in the evaluation process. The long-term goal is to streamline the process associated with the USE-ME conceptual framework to a point where it is more easily adoptable by language engineers.

8.5.3 Research question 2

In this section, we report on the statements related to RQ2 ' How can we promote usability concerns since an early stage of development of the DSL?' of this thesis. We state that **We can promote usability concerns since an early stage of DSL development by applying USE-ME conceptual framework** .

In Figure 8.39, we can see that 90.4% of participants Agree that the USE-ME conceptual framework is suitable to be applied from the early stage of the development of the DSL. Two participants reported that they do not have a strong opinion, while three Disagreed with a statement. Participants which disagree provided feedback on following additional question 'Can you please let us know why do you disagree with a statement "Approach is suitable to be applied from the early stage of the development of the DSL":' as follows:

- *For most of the assessment of usability, I expect you need an implementation of the DSL.*
- *I believe it's too complex. There might/should be easier methods to do a rather quick assessment. Otherwise, engineers will simply skip using the method.*
- *In my experience, the first version of a DSL is frequently developed without a well-defined process and without clear goals, thus the de facto process is not always as nice as in Fig. 3. I think that the presented approach is really beneficial once this early prototyping phase is reached to systematically evolve the DSL towards a usable one.*

In summary, the subjects doubt that it is possible to apply USE-ME approach during DSL development while there is no functional DSL prototype. However, we disagree with this, as the USE-ME conceptual framework can be used to obtain a relevant feedback in early phases of development. For instance during the decision and domain analysis phase, the survey with potential users can be organize in order to clarify the domain context and problems. Further on, the specification of the abstract and concrete syntax elements can be evaluated early, to find out the level of understandability or a readability of the same. Finally, the conceptual framework can be applied in parallel to DSL development from beginning, as it requires the context and goals definition, which can be obtained in

the early development of DSL and will shape the evaluation which can be executed when having the first prototype.

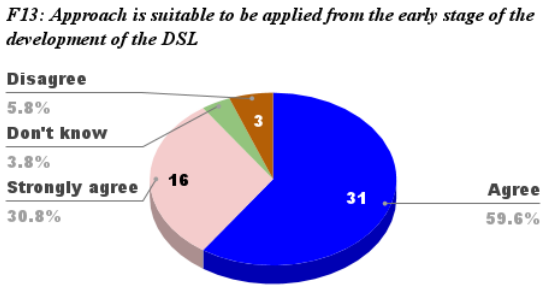


Figure 8.39: RQ2 feedback (FQ13)

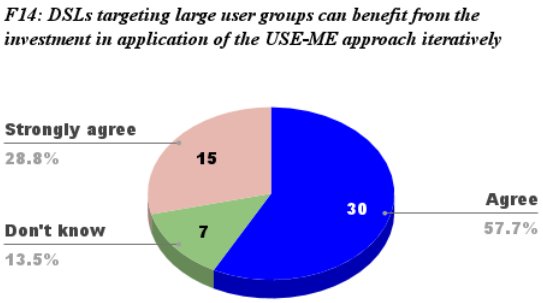


Figure 8.40: RQ2 feedback (F14)

Further, in Figure 8.40 we can see that 96.5% participants Agreed with the statement that DSLs targeting large user groups can benefit from the investment in the application of the USE-ME conceptual framework iteratively. However, 13.5% of participants reported not to have a strong opinion, while no one Disagreed with a statement.

8.5.4 Research question 3

In this section we report on the statements related to RQ3 'How can we integrate the proposed conceptual framework to build usability evaluation into the development process of the DSL?' of this thesis. We state that **We can integrate conceptual framework to build usability evaluation into the development process of the DSL by USE-ME tool support**.

In Figure 8.41, we can see that 82.7% of participants Agree that the USE-ME tool makes it feasible for a DSL engineer to model usability evaluation. The remaining 17.3% of participants reported staying indifferent. Further, in Figure 8.42, we can see that 68.8% of participant Agree that the USE-ME tool is expressive enough for specifying usability evaluation of DSL. On the other hand, 28.8% don't have a strong opinion, while one participant Disagreed with the statement. They provided feedback on following additional question 'Can you please let us know why do you disagree with a statement "The USE-ME tool is expressive enough for specifying usability evaluation of DSL":' as follows:

I was getting hard time to understand the meaning of all those notations that have been used. It might be because of having it as a prototype. I think a DSL designer might need to spend much more time on this part than developing the DSL. Maybe better expressions and simplification might help.

In Figure 8.43, we can see that 90.4% of participants Agree that the USE-ME tool supports the integration of usability evaluation approach into the development process of the DSL and 9.6% of participants reported to stay indifferent. Further, in Figure 8.42, we can see that 63.5% of participant Agree that investment into the development of the

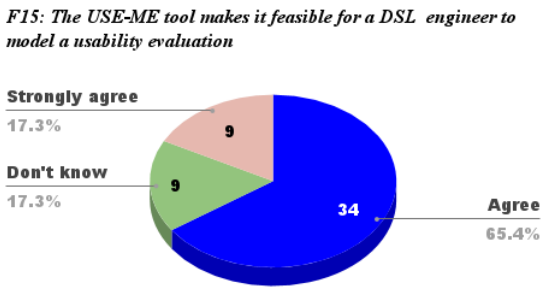


Figure 8.41: RQ3 feedback (FQ15)

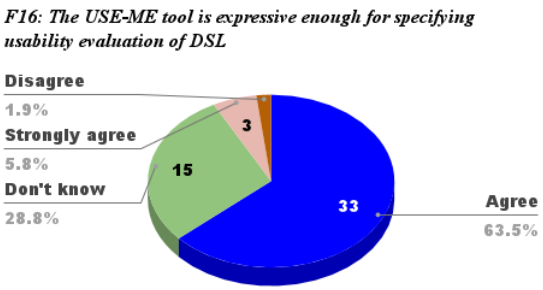


Figure 8.42: RQ3 feedback (FQ16)

USE-ME prototype tool into a real product is worthy. On another hand 36.5.8% don't have a strong opinion.

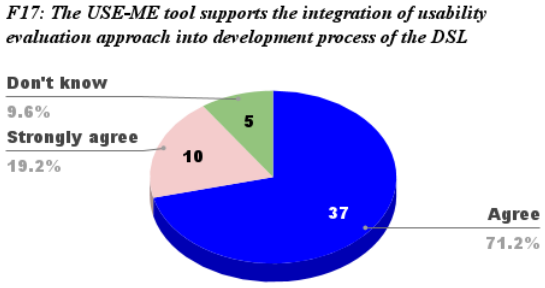


Figure 8.43: RQ3 feedback (FQ17)

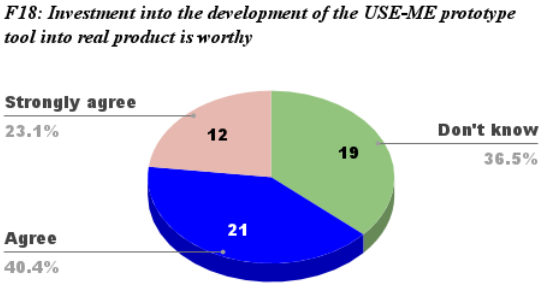


Figure 8.44: RQ3 feedback (FQ18)

8.5.5 Suggestions

We asked the survey participants to report their suggestions how to improve the USE-ME tool or systematic approach itself. Two participants provided their suggestions regarding the improvements on presentation materials and examples:

- *'The tutorial is hard to follow, diagrams are complex. Maybe a simpler example could help !'*
- *'More practical examples from the field with more complex scenario and examples of results. I think the scenario provided in the examples in the video was too general.'*

We received following two suggestions regarding general improvements on the tool:

- *'The ideas are really nice, however, I think the tool should be much leaner/simpler, boiling down DSL usability engineering to its essentials and also minimizing DSL eval. and improvement effort. Maybe a set of nicely crafted configurable checklists hiding part of the Eclipse modeling panes and using the same data model.'*

- *'The tools seems to be intuitive, I like it in general. Just on the surface of its understanding, I can say that you need to pay more attention how to visualize a global context of your approach, i.e. how to visually express a "current position"of a user in the global (i.e. methodology) context of your approach, and assist her or his in answering the questions: "Currently, where am I, and what are the next steps in my work?"To be more concrete with this my answer, I need to go deep in your tool and the approach.'*

One of participants suggested calculations of the cost effectiveness of USE-ME by providing the following comment: *'Investigate the link between the approach/tool on the one hand and the economic return on the other hand'*. Another two participants reported reducing the effort necessary to use the USE-ME by hiding unnecessary complexity:

- *'It strikes me as a bit too complex; perhaps it can be introduced gradually in an incremental fashion, starting with the core aspects first and then moving on to refinements of these in subsequent iterations. As it stands, I am concerned that it requires a lot of different types of background to use USE-ME effectively. (Mind you, this is based solely on watching the video, so I do not have much confidence in this recommendation of mine.)'*

- *'I would say, it's better to hide some tasks which are not really necessary or time consuming. If a method is going to be integrated to the development process, it should not take a lot of the developers time, otherwise, they will not use the approach. My main concerns is about the effort required for this process. Although, I know how it is important, but I may not use any method because of how time consuming they might be. If you tools can provide all benefits with less interaction from developers, it would be great. However, you approach might do it (but I have not tested it).'*

Finally, one participant reported on his doubts regarding the application of the USE-ME in early phases: *'"Altogether, I find this as a very systematic approach which aims to incorporate modeling best practices and advanced tools. My ""don't know""answers are dominantly emerging from past experience in observing the development of some real DSLs in the industry. In many cases, a couple of versions of the DSL have been developed as an early prototype, before starting to systematically develop a real DSL (with a well-founded development process). In my view, the presented approach can fit in at this stage - and maybe not at a very early stage of DSL development. At this stage, there is normally more information available about context, goals, etc - which can be totally missing in the beginning. I wonder if some ""default usability metrics""could just be collected (without a specific goal or context in mind) when a DSL starts to be developed in an ad hoc way. As such, the tool could raise a flag in case of major deviations in common usability metrics. These ad hoc metrics could nicely complement the systematically constructed usability evaluations. "'*. We agree that it is not always possible to apply the USE-ME conceptual framework early with final end users, but the assessments should be done with a people involved in the development. We showed in case of FlowSL development (Section 9.3, Annex III) that small assessments, performed early with domain expert, when systematically prepared can benefit the development

with early documentation examples and materials which can be reused when preparing more cost-full evaluations, for instance, empirical experiments with relevant number of participants.

8.5.6 Usability tools

In Figure 8.45 we can see that only four participants stated to be familiar with another tool which supports usability evaluations. However, two of them reported:

'Not exactly a tool, but some metrics to measure DSL quality (e.g., Moody's metrics)'

'I am familiar with theoretical methods for usability evaluation, but not tools per se.'

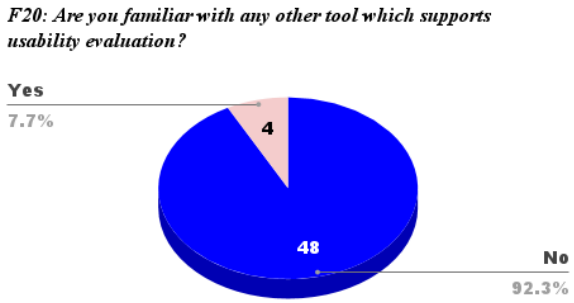


Figure 8.45: Feedback about other usability tools (FQ20)

Two other participants reported on the tool named Simpl and tools for testing web-pages (http://www.usefulusability.com/24-usability-testing-tools/). Both of participants Strongly agreed with following statements:

- The USE-ME is more suitable than alternative for usability evaluation of DSLs
- The USE-ME is more complete than alternative for usability evaluation of DSLs

8.6 Summary

8.6.1 Participants profile

In Section 8.4 we reported on the answers which we obtained to background questions of evaluation participants. We obtained 52 valid answers, from participants representing 21 different countries, and having obtained MSc and PhD degree. Over half of them had more than 10 years of research/work experience.

In Table 8.5 we summarize their modelling experience, which is a classifier for the USE-ME MDD Engineer profile, based on the self-reported feedback on their knowledge of Modelling (EQ4), MDD (EQ5), UML (EQ5) and UML diagrams (average of EQ7-10). In regard to this four characteristics, we estimated average modelling experience of each participant which indicates that 22.95% were experts, 45.55% advanced, 21.99% intermediate, 7.69% emerging and 1.8% without expertise.

Table 8.5: Participants modelling expertise

Modelling	Expert	Advanced	Intermediate	Emerging	No expertize
EQ4	28.8462%	42.3077%	19.2308%	9.6154%	0.0000%
EQ5	19.2308%	48.0769%	26.9231%	3.8462%	1.9231%
EQ6	23.0769%	46.1538%	19.2308%	9.6154%	1.9231%
AVG(EQ7-10)	20.6731%	45.6731%	22.5962%	7.6923%	3.3654%
Total AVG	**22.9567%**	**45.5529%**	**21.9952%**	**7.6923%**	**1.8029%**

Table 8.6: Participants DSL expertise

DSL	Expert	Advanced	Intermediate	Emerging	No expertize
EQ11	30.7692%	38.4615%	25.0000%	3.8462%	1.9231%
EQ12	23.0769%	25.0000%	38.4615%	3.8462%	9.6154%
Total AVG	**26.9231%**	**31.7308%**	**31.7308%**	**3.8462%**	**5.7692%**

Table 8.7: Other relevant participants background

	Expert	Advanced	Intermediate	Emerging	No expertize
AVG(EQ13-15)	9.6154%	23.0769%	26.2821%	16.0256%	25.0000%
EQ16	15.3846%	38.4615%	32.6923%	11.5385%	1.9231%
EQ17	3.8462%	13.4615%	38.4615%	26.9231%	17.3077%
EQ18	5.7692%	30.7692%	51.9231%	9.6154%	1.9231%

In Table 8.6 we report on participants experience with DSLs, which is classifier for USE-ME DSL Developer profile. We obtained average DSL experience based on participants general knowledge of DSL development (EQ11) and DSL in general (EQ12), as 26.92% being experts, 31.73% advanced, 31.73% intermediate, 3.85% emerging and 5.77% with no expertise. The participants which were not experienced with DSL development, were experienced in modelling or usability.

In Table 8.7 we summarize on participants other relevant experience. We obtained that over half of participants were experienced with the DSL modelling tools which were used for USE-ME development (average EQ13-15), however, the quarter of participants were not having any expertise with this tools. Also, half of the participants were experienced with goal modelling (EQ17). Over 85% of the participants had expertise in requirements engineering (EQ16) and agile development (EQ18). This indicates that more than half of the participants had enough background to easily comprehend the implementation and broather perspective of USE-ME conceptual framework (e.g. its iterative incremental application and information flow between existing DSL requirements and goal model).

In Table 8.8 we present relevant experience which impacts the usability classifier, which identifies USE-ME Usability Engineer profile. Based on the self-reported feedback on their knowledge of HCI (EQ19), UCD (E20), empirical experiments (EQ21) and usability testing (EQ22). In regard to this four characters, we estimated average usability experience of each participant which indicates that 4.81% were experts, 21.63% advanced,

Table 8.8: Participants usability experience

usability	Expert	Advanced	Intermediate	Emerging	No expertize
EQ19	5.7692%	19.2308%	40.3846%	25.0000%	9.6154%
EQ20	1.9231%	19.2308%	19.2308%	34.6154%	25.0000%
EQ21	7.6923%	36.5385%	25.0000%	19.2308%	11.5385%
EQ22	3.8462%	11.5385%	40.3846%	25.0000%	19.2308%
AVG	**4.8077%**	**21.6346%**	**31.2500%**	**25.9615%**	**16.3462%**

31.25% intermediate, 25.96% emerging and 16.35% without expertise.

In Figure 8.46 we summarize above reported experience regarding modelling, DSL and usability. If we take into account that ones which had intermediate, advanced or expert experience are having more than medium knowledge o each characteristic, we can calculate from given results that 90.5% of participants fit Modelling engineer profile, 90.38% DSL developer profile and 57.69% Usability engineer.

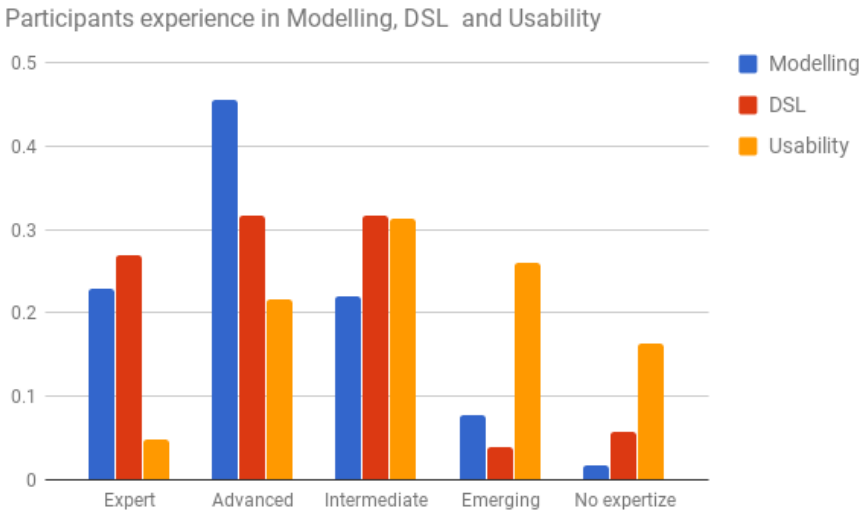


Figure 8.46: Participants experience in Modelling, DSL and Usability

Based on obtained background data, we conclude that all participants fit the USE-ME [End User] profile, and the majority of them had medium to high knowledge of modelling, usability or/and DSL development. Taking in consideration also the academic degree and working/research experience, we can note that we obtained feedback from the experienced target audience of USE-ME conceptual framework.

8.6.2 Research questions analysis

In this Section, we report on the results of our evaluation objectives, introduced in Section 8.2.2, which are meant to validate how successfully we addressed research objective of this thesis.

Table 8.9: Motivation feedback

Motivation	Agree	Indifferent	Disagree
FQ4	90.3846%	9.6154%	0.0000%
FQ5	96.1538%	1.9231%	1.9231%
FQ6	46.1538%	51.9231%	1.9231%
FQ7	96.1538%	3.8462%	0.0000%
FQ8	96.1538%	3.8462%	0.0000%
Avg	**85.0000%**	**14.2308%**	**0.7692%**

Table 8.10: RQ1 result summary

RQ1	Agree	Indifferent	Disagree
FQ9	92.3077%	7.6923%	0.0000%
FQ10	96.1538%	3.8462%	0.0000%
FQ11	53.8462%	28.8462%	17.3077%
FQ12	86.5385%	11.5385%	1.9231%
AVG	**82.2115%**	**12.9808%**	**4.8077%**

In Section 8.5.1 we reported on participants feedback on our motivation statements regarding the necessity of USE-ME conceptual framework and a need for the context-dependent and reusable usability evaluation approach for DSLs. We summarize the answers to each question as Agree (which includes Agree and Strongly Agree), Indifferent (for Don't know answers) and Disagree (for Disagree and Strongly Disagree). In Table 8.9 we summarize their answers and report on the score obtained an average of referring feedback questions (FQ4-FQ8). We can see that 85% of participants Agree with our motivation statements, 14.23% are indifferent, while 0.77% Disagree.

In Section 8.5.2 we reported individual results to questions related to evaluation goal G1, associated to RQ1 of this thesis, which answers to question *'Does USE-ME conceptual framework enable End Users to model DSL usability evaluations?*. In Table 8.10 we summarized the obtained feedback as the average of questions FQ9-FQ12, and we can see that 82.21% agree with our statements, 12.98% were indifferent and 4.81% disagree. Therefore we can conclude that:

Using [USE-ME conceptual framework] impacts the [feasibility] of [End Users] to model DSL usability evaluations.

In Section 8.5.3 we reported individual results to questions related to evaluation goal G2, associated to RQ2 of this thesis, which answers to question *'Does USE-ME conceptual framework enable End Users in promoting usability concerns since an early stage of DSL development?*. In Table 8.11 we summarized the obtained feedback as the average of questions FQ13 and FQ14. We can see that 88.46% agree with our statements, 8.65% were indifferent and 2.88% disagree. Therefore we can conclude that:

Using [USE-ME conceptual framework] impacts the [feasibility] of [EndUsers] to promote usability concerns since an early stage of DSL development.

In Section 8.5.4 we reported individual results to questions related to evaluation goal

Table 8.11: RQ2 result summary

RQ2	Agree	Indifferent	Disagree
FQ13	90.3846%	3.8462%	5.7692%
FQ14	86.5385%	13.4615%	0.0000%
AVG	88.4615%	8.6538%	2.8846%

Table 8.12: RQ3 result summary

RQ3	Agree	Indifferent	Disagree
FQ15	82.6923%	17.3077%	0.0000%
FQ16	69.2308%	28.8462%	1.9231%
FQ17	90.3846%	9.6154%	0.0000%
FQ18	63.4615%	36.5385%	0.0000%
AVG	**76.4423%**	**23.0769%**	**0.4808%**

G3, associated to RQ3 of this thesis, which answers to question *'Does USE-ME tool support enable End Users to built usability evaluation into the development process of the DSL?'*. In Table 8.12 we summarized the obtained feedback as the average of questions FQ9-FQ12, and we can see that 76.44% agree with our statements, 23.07% were indifferent and 0.48% disagree. Therefore we can conclude that:

Using [USE-ME conceptual framework] impacts the [feasibility] of [End Users] to built usability evaluation into the development process of the DSL.

Based on obtained feedback we can interpret that we succeeded to address research objective of this thesis (Section 1.2) by providing a USE-ME conceptual framework (Chapter 5) and associated tool support (Section 6.2). We obtained confirmation of experienced members of community that the USE-ME conceptual framework address well the research problems tackled in this thesis and that the topic is relevant for the research community.

8.7 Interview analysis

We collected data by an approach that combined interviews and survey. After participants filled in the evaluation survey (Section 8.3) during the pilot run, the thesis author conducted 1-hour long semi-structured interviews (in person or via Skype) with the following participants:

P1 Project owner from the FlowSL case study [23] (Annex III, Section 9.3) who was managing the development process. This participant holds PhD and has both industry and academic background, with more than 25 years of research experience. The participant reported on advance knowledge regarding modelling techniques and DSLs, and was an expert in agile development. However, the self-reported knowledge related to usability indicators was intermediate.

P2 Main developer from the Visualino case study (Annex IV, Section 9.4), with MSc

degree. This participant had 12 years of industry experience, and 8 years of research experience. S/he reported to be emerging regarding modelling experience and to have intermediate knowledge about DSLs and usability.

P3 Developer of the DSE merge approach (Section 9.5, Annex V) with three years of research experience and an MSc degree. The participant reported advanced knowledge of modelling and to be expert in DSL development, however, without knowledge of requirements engineering and emerging knowledge related to usability.

P4 Developer of the RDAL requirements approach which was merged with USE-ME [25] (Annex VI, Section 9.6), holding PhD degree and with 11 years of research experience and 10 years of industrial software development experience. The participant reported to be an expert in modelling, DSL development, and requirements engineering and having emerging knowledge related to usability.

P5 Evaluator of the SEA-ML [55, 56] which applied our systematic approach in its early phase, having both academic and industry experience and holding a PhD degree. The participant has 7 years of research experience and being advanced in modelling and DSL development, and emerging in requirements engineering and usability.

All participants strongly agree that usability evaluations are necessary for a DSL development in practice (FQ5) and that DSL evolution cycle should include usability re-evaluation (FQ8), while they agreed or stay indifferent with the other motivation statements. Regarding the feedback questions related to research questions (FQ9-FQ8) of the thesis the participants agreed or strongly agreed with most of the statements, having one indifferent regarding questions FQ14, FQ16 and FQ18. Finally, all interview participants reported not to be familiar with any other tool which supports usability evaluations.

During the interview session, all participants reported to find the video a bit too long, but very comprehensive. Participant P1 reported *'Video was complex, but it is very well organized and it was easy to follow the process. Quality of video is very good'*, indicating that the complexity is naturally inherited from the complexity of problem which the USE-ME conceptual framework captures. Further participant P5 reported *'Video material was very good, with paper I couldn't imagine how the tool would work'*, pointing out that the video material illustrates the process in a more pragmatic way. Participants further suggested breaking the video in smaller tutorial videos in order to enable the users to apply usability evaluation approach.

Regarding the USE-ME conceptual framework, the participants reported to like it, and one of them pointed that it could be interesting to distribute it to industrial context as a service. Regarding the applicability of systematic approach, participant P3 pointed *'I can not know, but I think it can be cost-effective to apply usability evaluation approach iteratively as we can reuse many things from previous evaluation if the changes were not too big'*. On the other hand, participant P5 noted that *It's risky to apply the process, it is dependent on how much people are developing DSLs and how big is the target market'*. We agree that for the

DSLs which are developed just for small teams, it may not be so beneficial to apply the approach, as it targets the large user groups, especially cases where users are not involved in the development or lack the programming background.

Finally, we asked participants about their estimate of how feasible it would be to prepare the evaluation process following the USE-ME conceptual framework if they had it while performing the evaluation of their DSLs. Participant P1 reported *'Having the conceptual framework would be beneficial for the project, we needed the guidelines'*. Participant P2 pointed *'I would have a basis to use the approach with USE-ME conceptual framework. It would be interesting to have proper tests along the project. We never kept track of timing and were not having formal questionnaires. Ideally, we would apply the approach like it is'*, indicating that if the evaluation assessments were modelled, things will be clear and the team included in the organization of experimental assessments would be more in sync with preparation and execution. Participant P3 pointed *'Well, yes, USE-ME conceptual framework could help me to prepare evaluations alone. However, I learned a lot about it with you when we were conducting the experiment so I can not tell if someone could do it just by following the guidelines'*. Further, participant P4 reported that *I would need to look examples and process in order to do it alone. However, it would it be difficult to create it from scratch with a tool without having guidelines, for instance using CheetSheets'*. Finally, participant P5 who already performed an evaluation by himself reported *'If I had the approach I would do better, it would help me exactly with things I missed in my evaluation. I focused on the evaluation of how complex is to create, and how generative DSL is, from the perspective of language engineering, focusing on performance. I found USE-ME nice because approach consider different users and their perspectives'*, indicating that s/he would like to make evaluation using the prototype tool, finding it very comprehensive, but doubting that novice developer could grasp it at the first time.

8.8 Threats to validity

The results presented in this Chapter are a good indicator that the USE-ME conceptual framework supports quality in use of DSLs during their development process by leveraging usability as a first-class concern. In this Section, we challenge the conclusions drawn from the study by indicating threats to validity and how these threats affect the study.

One of the internal threats to the validity of this study is that the *participants did not use the approach in practice*. We performed a small evaluation study with few novice users which were included in the development of the different DSLs in the context of DSL graduate course. The study did show that they managed to prepare evaluations using the tool. For our final evaluation, presented in this Chapter, we performed the evaluation with experienced people in the DSL research domain. The available time of this people is restricted, and therefore it would not be possible to obtain their feedback if we asked them to invest a lot of time participating in a controlled experiment. To lower this threat we provided the video tutorial which presents USE-ME conceptual framework

in a comprehensive way. Based on the feedback regarding the video during the interview session, we could observe that participants could understand the motivation and the application of the USE-ME in this 15min presentation. All participants were provided also with the detailed explanation of approach published in COMLAN journal the [26] and the prototype tool with the example models.

Another internal threat to the validity of our evaluation study is the *respondent bias*, as the subjects sometimes may report what evaluator would like to hear, especially during the interview sessions. However, we included in the study the participants with different backgrounds, having different maturity level in the area. None of the participants had a special interest in the results of the study. We obtained homogeneous results to our statements, and therefore we believe that we succeed motivate the participants to focus on the importance of the problem.

Further, the internal threat regarding the interview process is the *researcher bias*, as the researcher could misunderstood what the participants were reporting during the interview sessions. Therefore, the report introduced in Section 8.7 was sent to participants, so they can confirm that the reported interpretation is correct.

Finally, regarding the external validity of the study, e.g. that conclusions from this study can be generalized to other DSL development situations, we modeled the framework in a way that it can be adopted in different situations, which we discussed in Chapter 7. A majority of our participants had relevant experience in DSL development, indicating their experience in different domains and different development processes. Their answers indicate that they perceive the approach applicable in each of their cases.

Based on obtained feedback we can interpret that we succeeded to address research objective of this thesis (Section 1.2) by providing a USE-ME conceptual framework (Chapter 5) and associated tool support (Section 6.2). We obtained confirmation of experienced members of community that the USE-ME conceptual framework address well the research problems tackled in this thesis and that the topic is relevant for the research community.

Chapter 9

Case studies

We conducted several case studies (see Figure 1.1) to illustrate the proposed USE-ME conceptual framework;

- We introduce in Section 9.1, 9.2 and 9.5 three academic case studies [15, 18, 141] which were used to put into practice both the evaluation model and experiment design, proposed in Chapter 4, and served to get introduced with DSLs' development life cycle.

- In Sections 9.3 and 9.4, we introduce two industrial case studies which were used to apply the approach during their iterative development (FlowSL and Visualino DSL). In both cases, the researcher did not take part of the DSL development, but was taking a role of expert evaluator. More particularly, it was possible to observe with FlowSL that early evaluations were beneficial and easy to integrate with the agile development process. On the other hand, the experience with Visualino, where the controlled experiments were carried on each release, put into evidence the reuse of the evaluation model instances, besides significant improvements in usability.

- In Section 9.6 we introduce work done to integrate Requirements Engineering tools in the DSL development process in articulation with the USE-ME implementation prototype [25].

9.1 Pheasant

To illustrate the experimental model for usability evaluation referred in Chapter 4, we took an existing DSL for HEP called Pheasant. Pheasant is a project held in the context of HEP Physics. It aimed at developing a Domain Specific Visual Query Language to

provide the end users (Physicists) with a tool for mining Physics Data stored by the large detectors during the provoked the particle collisions [7].

The DSL on this case study published in [18] (see Annex I) was used as a first usability evaluation example during the problem investigation phase of our research process (Figure 1.1). It also served as illustration example for patterns (Appendix B) of pattern language which instantiates our design model (Section 4.2).

9.1.1 Purpose of Pheasant development

The Pheasant project was developed to mitigate users productivity problems in the context of HEP domain. It aimed to develop reusable engineering methodologies through MDD techniques. A declarative Domain-Specific Visual Query Language was used to raise the abstraction level in the existing query systems and give room to new optimization of different levels. The goal of Pheasant was to automate this process as much as possible, as well as to provide the physicists (with profiles ranging from the ones without programming expertise to high-level programmers) appropriate abstractions that hide the complexity of programming error-prone algorithms in languages (e.g. C, C++ or Fortran), by using a wide plethora of libraries and frameworks to achieve their goals.

The Pheasant development and detailed data analysis were part of the before-mentioned PhD thesis [7], while we performed the statistical analysis and used this detailed data for illustration purpose of usability evaluation of DSLs. The performed work served to confirm that the proposed query DSL tailored to the specific domain was beneficial to the End User. The physicists, non-experts in programming, no longer were required to cope with different GPLs and adapt to the intricacies supporting database infrastructure.

9.1.2 Pheasant usability evaluation

The evaluation process followed in this case study is presented in Figure 9.1. The process starts with the Participants Recruitment, where the users are analyzed and grouped into clear categories. This way, the variables concerning the user profile that lead to different results for different groups are controlled. This step is followed by the Task Preparation. The aim here is to organize the evaluation by determining which tasks have to be done and which tests are elaborated in order to provide the proper results. This will generate the information required to be analyzed afterwards. The next step is the Pilot Session, which is meant to simulate the exam and test that the material for the training and the evaluation procedures is well organized. The main advantage of this rehearsal is to check that the time constraints and other possible external variables like proper equipment are controlled, and do not interfere with the results. Once everything is tested, we proceed to on the assessment, which we call Evaluation Session, for each group and language being compared. A Training Session is used to introduce the language. At this stage, Immediate Comprehension and Review tests are conducted with participants, while introducing the language features. The final exams, in the Exam Session, involve sentence writing

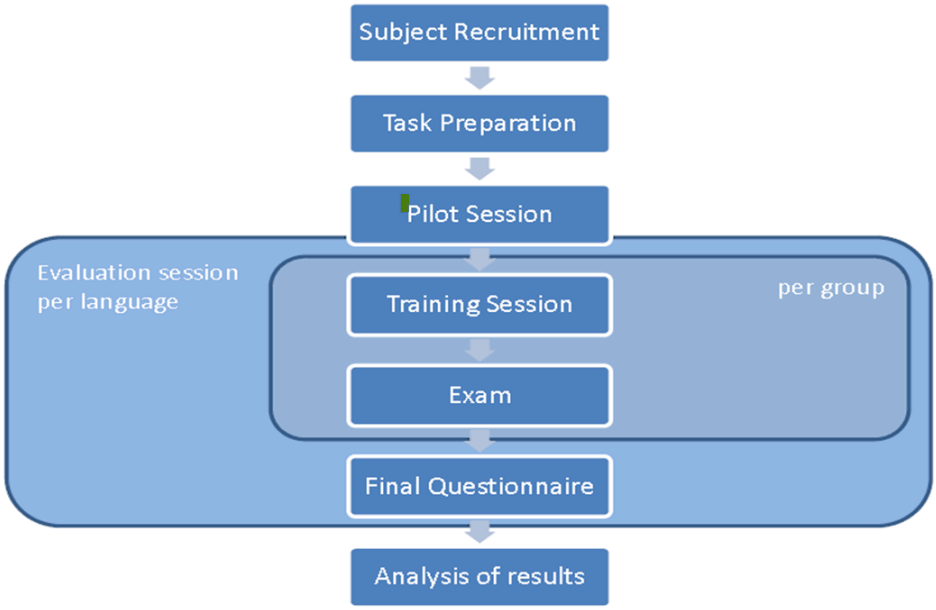


Figure 9.1: Evaluation process for Pheasant (taken from [18], based on [7])

activities. During the exam session, participants' activities are observed and recorded, so that information such as completion times and error rates can be collected. The goal is to determine the ease of learning. After each group has been evaluated in the different languages, the participants are asked for a debriefing in the form of a Final Questionnaire Session. The goal is to obtain the user's qualitative perspective of the comparison between the languages. In order to evaluate unbiasedly, the users should test the same environment and as realistically as possible. Evaluation process terminates with the Analysis of Results.

Our usability assessment includes Physicists with programming experience with two profiles: the ones with no experience with the previous framework used and the experienced ones. The goal was to analyse the performance of Pheasant programmers compared to the baseline alternative with respect to the efficiency, effectiveness and confidence in defining queries in Pheasant. The assessment was done from the point of view of a researcher trying to assess the Pheasant DSL, in the context of a case study on selected queries.

Introducing one language to the whole group of participants and only afterwards the other language would bias the evaluation, as the knowledge acquired while learning the first language could be partially reused while using the second language. To mitigate this threat to the validity of the results we had to split the group in two. This way, it reduced the influence of the first language while presenting the second. Mixing the two groups might lead to new variables in the evaluation that are hard to track. Therefore, it was necessary to organise four sessions, with each group taking part in two sessions (one for each language). Following the scientific method, the participants' performance in the

query writing was evaluated. Every participant had four queries, specified in English, to be rewritten in the previously learned language.

Using Pheasant, the users increased effectiveness during their query specification. The DSL was less error-prone than the alternative; it allowed non-programmers to define their queries correctly. The evaluation also showed a considerable speedup in the query definition by all the groups of users that were using Pheasant. In general, the feedback obtained from the users was that it is more comfortable to use Pheasant with the alternative. The preliminary pilot study was fundamental to ensure that the subject's time was well spent.

9.1.3 Conclusion of the case study

This work's contribution illustrates how an experimental process (Section 4.1) can be used in the context of a DSL evaluation, with respect to its impact on Quality in Use characteristics (Section 2.2), namely effectiveness and efficiency. The valuable feedback from users concerning the tool support for the language, as well as their fears concerning language expressiveness, support the idea of an iterative evaluation process where improvements to the language and its tool support are to be performed and then assessed in a new round of evaluation.

9.2 RPG DSL

In this case study, an RPG DSL for product lines was developed, which was completely built using MDD software development techniques [141] (see Annex II) and served as DSL development example during the problem investigation phase of our research process (Figure 1.1). We have shown several benefits of applying MDD to DSL development regarding prototyping of cross-platform games, and their evaluation by means of static and dynamic verification techniques of the game's logic properties.

9.2.1 RPG DSL development

In the Domain Analysis phase, we worked on two different levels: at the problem level (i.e., expressing the concepts and logic of RPG Games), and at the solution level (i.e., how those concepts can be realised in a computational platform). At the level of the problem of RPGs' design, we tried to express and define what would be the common characteristics of all RPG games, regardless of what are the requirements of their implementation on an underlying computation infrastructure. At the solution (computational) level, we had to choose platforms to deploy the generated games and to analyse them. We chose a game developing platform and built our framework on top of it. The criteria to select the target framework from the existing game engines were: fast development; provided abstraction level relatively to system calls and hardware dependencies (e.g., graphical primitives, input modalities, etc.); and, need to previous knowledge in the area of game engines.

We chose Corona SDK 2 from a set of three promising candidate frameworks because it supports the possibility to compile the game for different mobile platforms. The game is programmed using the Lua language: a scripting language, which is preferred for rapid development. The creation of an API over the framework allowed a model to model transformation that was easily produced between RPG and U-Framework meta-models, which consists mostly of 1 to 1 relationships. This strategy of bottom-up modelling in the framework allowed the focusing on the RPG entities and the restriction of the power of the framework which was very low-level. The mapping of the RPG meta-model to Algebraic Petri-net (APN) [128] is too much complex to simply perform it in just one step, therefore we created an intermediate meta-model U-RPG that allowed a simpler mapping between both. We believe that the use of intermediate languages really help in this process since we do not have to map complex entities directly to an APN .

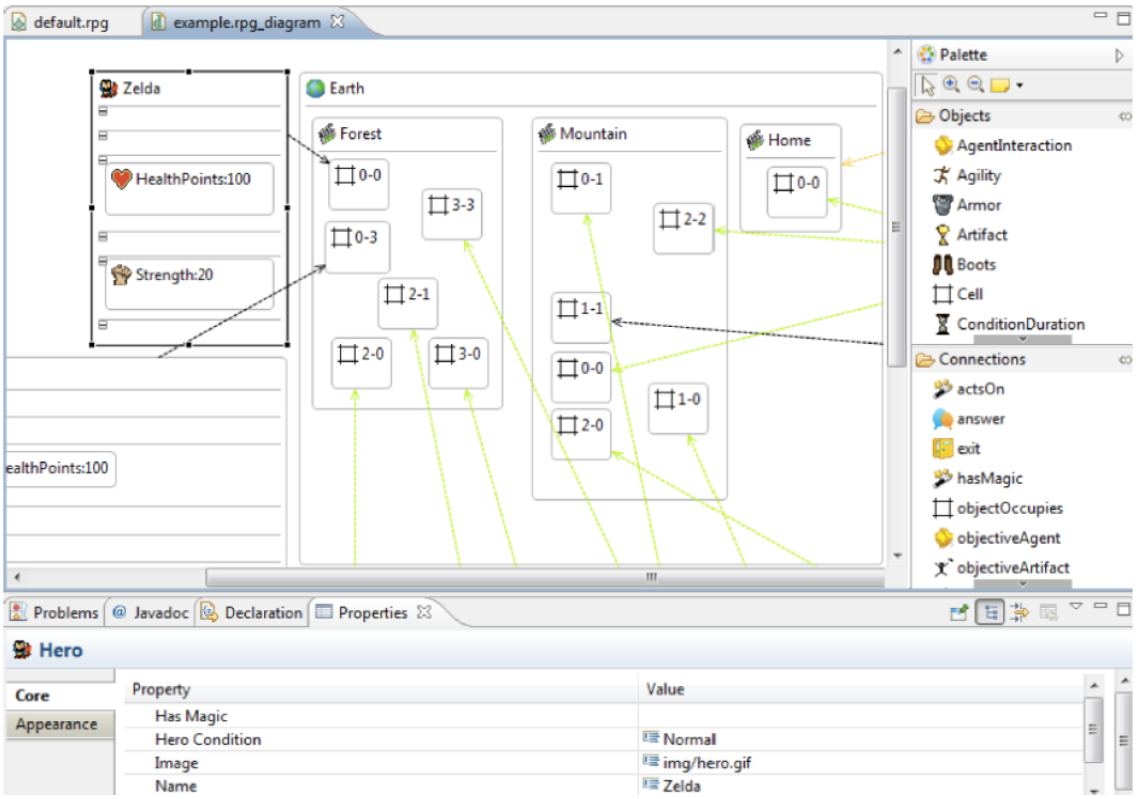


Figure 9.2: RPG DSL editor (taken from [141])

To complete the MDD cycle, we use a model checker to analyse and validate properties on RPG games, giving us a certain level of confidence about its implementation, since it passed verification phase. Regarding the RPG meta-model evolution, the addition of a new feature should not affect the existing ones if its concept does not interfere with existing features. However, if it does interfere, we have to analyze the impact of that interference in the U-RPG Meta-model, which may lead to a partial redefinition of these model.

The choice of the DSL concrete syntax (textual or graphical) impacted the process of

game development. A textual DSL may lead to a more readable solution than a graphical one in the development of big games since a visual one will have problems displaying all the information about the game. However, a graphical DSL (Figure 9.2) allows the developer to get a preview how the things will be mapped, allowing the game developer detect errors faster.

9.2.2 Conclusion of the case study

This study contributed to our decision to use the MDD approach for USE-ME development, as it showed to be beneficial in lowering the complexity of the underneath domain. For the domain of digital games, it showed to contribute to code reuse and facilitates game's verification. Application of the MDD is expected to increase productivity in the development (evolution and maintenance) of our engineering solution.

9.3 FlowSL

We applied action research to the development of a DSL, named FlowSL, designed to support managers when specifying and controlling the business processes supporting humanitarian campaigns [23] (see Annex III). Work was developed under the collaboration of the Engineering Faculty of Porto (FEUP) and Public Service International (PSI). This case study help us validate solution design which address the engineering problem of this thesis and is proposed in Chapter 4 as a part of our research process (Figure 1.1). Final report of this project can be downloaded from public repository [1].

9.3.1 FlowSL development

FlowSL is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organization PSI and is integrated into Movercado (MVC) platform[2]. This project consists of a mobile-based messaging platform at the core of an ecosystem that enables real-time and a more efficient impact, by facilitating interactions among beneficiaries, health workers and facilities, e-money and mobile operators. A first version of the system (MVC1) was developed as a proof-of-concept to validate the key underlying principles. The second version of the system (MVC2) was developed in the form of a platform easily customizable by managers and extensible by developers of the organization's team. An important goal was to develop a language, FlowSL, to empower the Campaign Managers to define new kinds of campaign flows taking advantage of their domain knowledge.

To balance the development effort with effective reusability (e.g. while envisioning new marketing solutions), MVC2 was developed in a fast-paced way, iteratively, along

[1] goo.gl/gQzX7B (accessed September 19, 2017)
[2] https://movercado.wordpress.com/ (accessed September 19, 2017)

six two-weeks sprints, following an agile development process based on Scrum [3] and best practices of evolving reusable software systems. In the process of development, the Domain Experts were part of the Product Owners team, while the Language Engineers were part of the Scrum Team. The DSL evaluation process was guided by the FlowSL development stages, as a different effort was estimated in each sprint for its development. The problem analysis was performed by mutual interaction and brainstorming between Domain Experts and Language Engineers in each sprint planning. We had the role of observing and guiding the analysis outputs, while preparing the evaluation plan, without being directly involved in the language specification.

To better understand and define the problem, the required functionalities were described in terms of small user stories. Also, the new description of the user roles was introduced as the FlowSL is expected to change existing organisational workflows. To improve interaction between the development team and the users, all the produced results from the analysis were continuously documented in a Wiki[4]. As Scrum suggests, the project management was based on a product backlog maintained and shared online.

The relationship between the MVC system, FlowSL development, and relevant language users and expected workflow is presented in Figure 9.3. The original MVC1 system was developed in a GPL (Ruby[5]), and naturally, FlowSL was first developed as a Ruby-based internal DSL. This approach allowed an optimal use of resources while keeping the existing system running. The second phase of language development was intended to support the managers to design the campaign flow specifications by themselves, using simple and understandable visual language constructs. In the third phase, the focus was on evolving the language's editor to be collaborative and web-based.

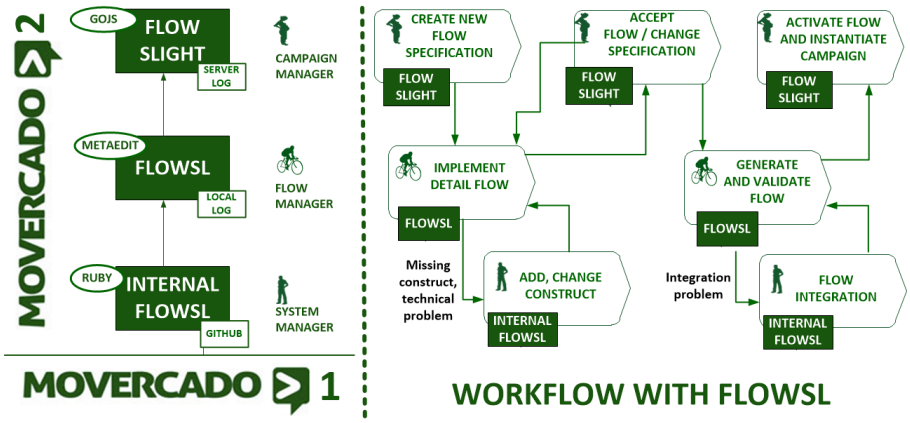


Figure 9.3: FlowSL workflow (Taken from:[23])

[3] http://www.scrum.org/ (accessed September 19, 2017)
[4] https://www.mediawiki.org/wiki/MediaWiki (accessed September 19, 2017)
[5] http://rubyonrails.org/ (accessed September 19, 2017)

9.3.2 FlowSL usability assessment

FlowSL is targeted to non-programmers. Their ability to use this language was identified as one of the highest concerns, so discovering usability issues in early development iterations facilitated the achievement of an acceptable usability, while tracking the design decisions and their impact. Usability has two complementary roles in design: as an attribute that must be designed into the product, and as the highest level quality objective which should be the overall objective of design.

Internal Ruby based FlowSL was defined as a first step to making distinction between the campaign implementations and a flow underneath, which helps to define the first draft of the concrete visual syntax of FlowSL, but was based on textual syntax. It helped to place initial definitions of flow concepts as well as target users. Evaluation goal was to assess whether this representation would be good enough to enhance the *understandability* and *readability* of flows from the perspective of Campaign Managers. It was expected that with the flow abstraction, the Domain Experts could describe more concrete requirements for the visual flow concepts. The *evaluation intervention* was conducted as a continuous interview with the Domain Expert with the role of Campaign Manager that was involved in specifying flows using the MVC1 system and who was also involved in the MVC2 Scrum development assuming, in that case, the role of Product Owner. The performed evaluation helped the DSL developers to adjust the level of abstraction to the needs of the DSL end users. The language at this phase could be used by the System Managers (knowledgeable of the concepts of the baseline system), but not by Campaign Managers.

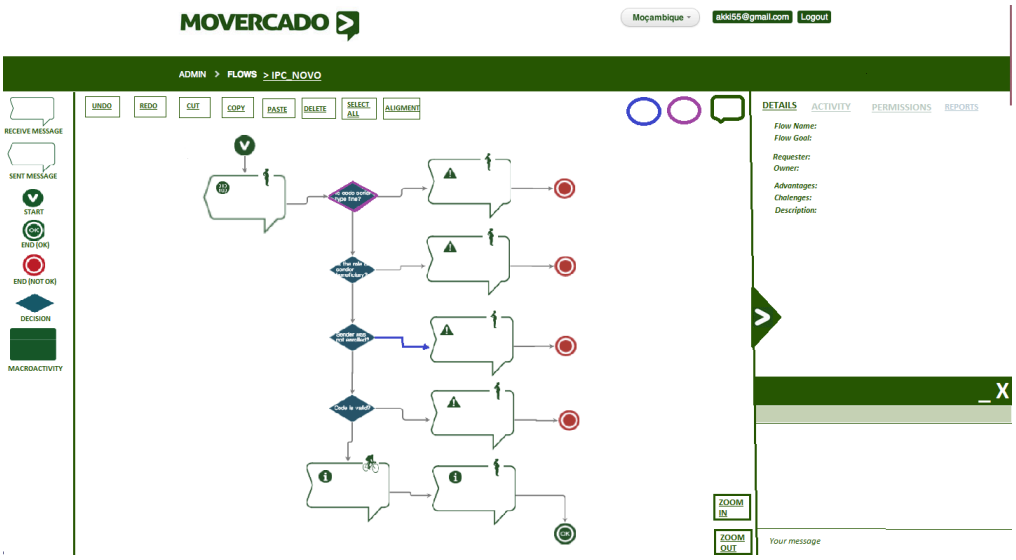


Figure 9.4: FlowSLLight interface integrated in MVC online platform

The of the second iteration was to develop a visual FlowSL prototype using the MetaEdit[6] language workbench, that was selected for its support to top-down development. The *evaluation's goal* was to assess whether both the campaign managers and

[6]http://www.metacase.com/ (accessed September 19, 2017)

novice system managers were able to validate the specified flows using the newly proposed visual language and editor. These evaluations covered also the effectiveness and expressiveness of the target language. The *First evaluation intervention* was organized very quickly and involved interviewing two *subjects*: the campaign manager and the system manager, both were involved in the DSL development. The *second evaluation intervention* involved the same subjects and focused in assessing the *understandability* and *expressiveness* of the individual symbols and to measure the *readability* and *efficiency* of the designed solution of the simple and complex flow. For the *third evaluation intervention* the usability engineer introduced the design improvements motivated by the feedback obtained the previous evaluation. The new notations were designed and implemented, to be again compared. The tasks were similar to the previous intervention, although more elaborated. Here, the same subjects from the previous interventions were involved, as well as a member of the Scrum team.

It became clear that the evaluation materials prepared earlier helped to speed up the following evaluation phases and reduced their implementation costs. Besides, they became templates for the corresponding learning materials. Also, it was possible to abstract the language one level further, so that an online visual editor was built to support rapid high-level specifications of flows. To better deal with the increasing complexity of the specified models, rather than presenting all the concepts related to the flow definition visually, a better option would be to present just high-level concepts that are reused often, while others are hidden and based on predefined rules that can be eventually reconfigured textually. This approach empowered both the domain experts and the product owners to better control the design decisions.

We integrated top-down usability engineering practices into a bottom-up agile development of the FlowSL from its beginning. While playing the role of Evaluation Expert that is expected to be introduced into DSL development, we experienced that the small iterations involving other project stakeholders than Language Engineers, namely Domain Experts, Product Owners and End Users, can help us to clarify the meaning and the definition of the relevant language concepts. This enables an early identification of possible language usability shortcomings and helps to reshape the DSL accordingly. Early evaluations can be executed with a relatively low-cost thanks to model-driven tools (i.e. language workbenches) that support the production of rapid prototypes and presenting the idea. These evaluations support well-informed trade-offs among the strategy and design of the DSL under development, and its technical implementation, by improving communication. Besides, they improve the traceability of decisions, and of the solution progress. These iterations also help to capture and clarify contractual details of the most relevant language aspects that need to be considered during DSL development and are an essential element to improve the End Users experience while working with FlowSL.

Finally, FlowSLLight was developed using GOJS tool as a web editor that is to be integrated into MVC platform web interface. This solution, as recommended, was built to support just light specifications and changes on the campaign flow. It was found to be

good enough to enable also visual campaign instantiations and simulations. Its purpose was to enable collaborative approach in specifying and validating the flows. The design ideas were evaluated and discussed with managers, regarding its use and integration into the platform. The final design of the editor in the MVC web interfaced was proposed and accepted (see Figure 9.4).

9.3.3 Conclusion of case study

This study helped us to apply our preliminary usability evaluation approach described in a form of pattern language (Chapter 4, Appendix B) and to perform the different usability evaluation methods (e.g. interviews, observations, heuristics) (Appendix A) on unfinished DSL. Weaving usability concerns into the agile development process helped to continuously evolve FlowSL, improving the cost-effectiveness of DSL usage in specifying campaigns, and supporting a clearer assessment of which language concepts are more relevant to the different kinds of language users, which in turn helps to find the right level of abstraction and granularity of concepts. All these benefits come with the cost of adding usability skills and of introducing new practices in the agile process, namely the introduction of lightweight meta-modelling tools.

9.4 Visualino

In this Section we include the unpublished article about the evaluation of the DSL Visualino [29] (see Annex IV), for the programming low-cost robots. The work was developed under the collaboration between the group ASE NOVA/LINCS and Artica[7], a company that specialises in the development of robotic and audio-visual solutions.

The first language design was developed in 2013 in the context of the master thesis [135], where language was named Farrusco. Later, the language continued to evolve and was renamed to Visualino after the second release in 2015, and recently Gyro, after the third release in 2016. We managed to show that the usability of the Gyro significantly improved in terms of efficiency and satisfaction when compared to an earlier version which detailed in experiment repository[8].

This study presents the application of our usability evaluation approach (Section 4.2) to the industrial case study during several development iterations. Also, it served as a design validation example of our research process (Figure 1.1). Further, it was used to validate feasibility of our conceptual framework (Chapter 5) as a USE-ME model instantiation example (Section 6.2). Further, it served as a running example for our integration study (Section 9.6, Annex VI).

[7] http://artica.cc/ (accessed September 19, 2017)
[8] https://sites.google.com/view/vl-empiricalstudy/home (accessed September 19, 2017)

9.4.1 Visualino development

The focus of the Visualino development was to build a dedicated DSL that removes the programming details, to control a low-cost Arduino rover robot[9], and at the same time allows children to easily get acquainted with it and preserve the possibility to program complex behaviour. A visual programming language allows the user to implement programs, through the manipulation of visual elements or objects. This manipulation is deduced in a visual way, allowing the user to understand programming mechanisms quickly, increasing their accessibility to new systems. Users with no programming background may implement programs in a simpler form. Visual languages introduce programming concepts to children, while robots perform the developed code in the real world. Children have the opportunity to observe their own developed program, running on a physical robot.

The chosen meta-modelling workbench for the DSL implementation was Eugenia for Eclipse. Its first development iteration was a part of the master thesis, where its abstract syntax and initial concrete syntax was proposed [135]. The DSL implementation started with domain analysis that brought concepts and details, so it was possible to develop the DSL meta-model. Among different behaviour paradigms for visual language, after extended analysis, it was chosen to represent a language through behaviour trees. This last paradigm is an alternative to the state machines and comprehends a hierarchy of behaviours, with an objective to fulfil. Each node may have a specification that determines how the actions of its children will be executed, which may be in parallel or sequentially.

Visualinos visual syntax is based on the behaviour tree paradigm [137], a mathematical model of plan execution, used for a diversity of areas including robotics, control systems and software games. A complex behaviour is mapped into smaller and simpler behaviours through its branches. This descending order of complexity provides a structured way of defining complex behaviours (which are used to define objectives) through simple tasks defined hierarchically. Each node may have a specification that determines how the actions of its children will be executed (in parallel, or sequentially). The child node returns its status to the parent node, and this successively happens until the root of the tree.

Fig. 9.5 illustrates how to program the robot to move back and forth. The root node is decomposed into a single sequential node (highlighted with a red circle). The sequential node runs all its children (in this case, the leaf nodes) in depth-first traversal order. Leaf nodes represent the most primitive actions that could be taken by an agent. In Fig. 9.5 the nodes represented by arrows pointing downwards are outputs actions. In this case, they are left and right motors command. The outputs can also be visual using LEDs or audio using a buzzer. Both nodes displayed by 2 round timer shapes are wait commands in which the execution of the next node is delayed by the duration determined by the end user for each node.

[9] http://www.arduino.cc/ (accessed September 19, 2017)

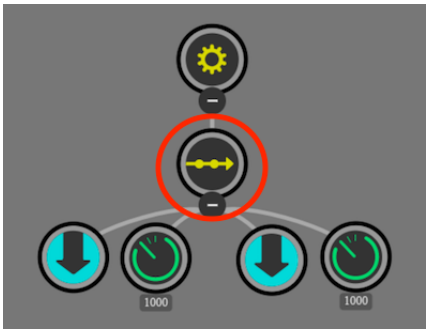


Figure 9.5: VL1 - Back and Foward

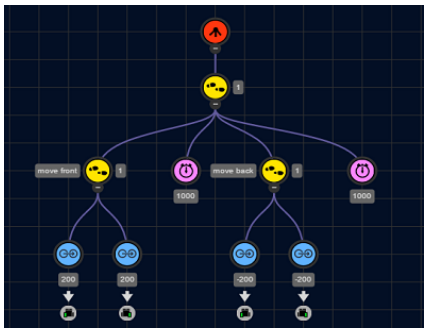


Figure 9.6: VL2 - Back and Foward with bumpers

The first development iteration of *Visualino* VL0 started with a domain analysis, followed by the design, implementation, and evaluation, in order to quickly deploy an early prototype. The last phase involved 22 (10+12) children, with the age range from 8 to 12, as subjects included in an exploratory study [135]. Children under 10 had a hard time learning *Visualino*. Further development resulted in the next release VL1 (Fig. 9.5), which improved the interaction model and provided a web-based solution.

VL1 was evaluated experimentally and compared to the one of the most popular commercial competitors in the market, *LegoMS*. Based on the results of this empirical study, some improvements were suggested and applied to Visualino development. The focus was on improving the user interface providing better readability of the programs being developed and improving error prevention.

9.4.2 Visualino usability evaluation

Visualino is developed iteratively reusing UCD methods. Programming technologies of rover robots for children still are excluding the children as target audience as it is hard for them to program in a textual language with a complex syntax full of technical concepts. To design an appropriate DSL for children is far from being a trivial task. There is no unique profile, and several factors related mostly to age, like the maturity level, that can influence widely in the design.

The children feedback is used to help to steer the Visualino evolution through the identification of several improvement opportunities in the language.This evaluation stage in each iteration is often not reported in the context of developing DSLs but is key to our development effort. In this assessment, we contrast Visualino with two popular DSLs that are used to control rover-like robots: a commercial competitor (Lego) and an open source initiative (Scratch).

We reported on the design and results of the empirical studies used in this evaluation that helped us identify the language's strengths and weaknesses. To help to achieve this higher level goal, we answered following two research questions:

- ***RQ1:*** *How does the current Visualino (VL2) compared to baselines (a previous version*

*of Visualino (VL1), Lego and Scratch) regarding the **Effectiveness** of the teenagers when programming a robot?*

- ***RQ2:** How does the current Visualino (VL2) compared to baselines (a previous version of Visualino (VL1), Lego and Scratch) regarding the **Satisfaction** of the teenagers when programming a robot?*

After the second development cycle, we organized empirical study to compare a second Visualino release (V1) with the well-known Lego[10] commercial DSL which paradigm is based on the building blocks composition. This evaluation helped to understand which features are missing or can be improved to make Visualino competitive to the best product in the market with the same purpose. After the third development cycle, compared the third Visualino release (V2) to the existing low-cost alternative MBlock[11].

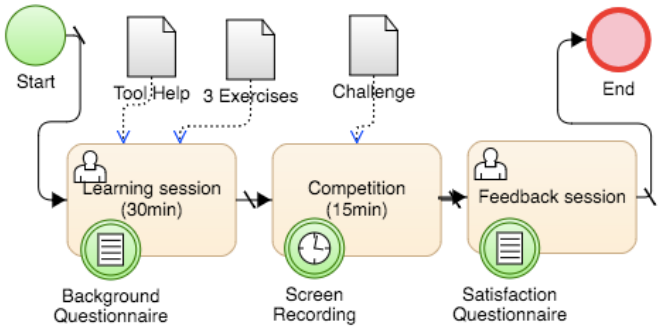


Figure 9.7: Experiment Flow

During the experimental studies, we used survey forms, video recordings and a competition arena to collect data for further analysis in different experimental sessions (see Figure 9.7). The survey forms were composed of "*Smileyometers*" which are found to be appropriate for teenagers questionnaires [181]. While answering to forms, children were assisted by an adult (one of the experiment assistants), to ensure that there were no misinterpretations of questions and answers and to confirm that participants did not experience reading problems. As we grouped the participants into teams, the participants' individual answers to questionnaires were merged.

We captured participants **Profile** as a measure influenced by *Experience* factors on Computer Games, Programming or Programming a robot, and *Tendency* factors reflecting the children tendency to Mathematics, Physics or to Learn programming. We used the recorded videos to evaluate the **Effectiveness**. The challenge could be solved by composing elements of the training exercises, which are marked as Success (S) or Failure (F). Effectiveness measures the percentage of modelling elements correctly built and composed to achieve the solution. **Satisfaction** was characterized by a *Confidence* factor which indicates tells how confident participants were about their solution; *Likeability* factor, that

[10] https://www.lego.com/en-us/mindstorms/ (accessed September 19, 2017)
[11] http://www.mblock.cc/ (accessed September 19, 2017)

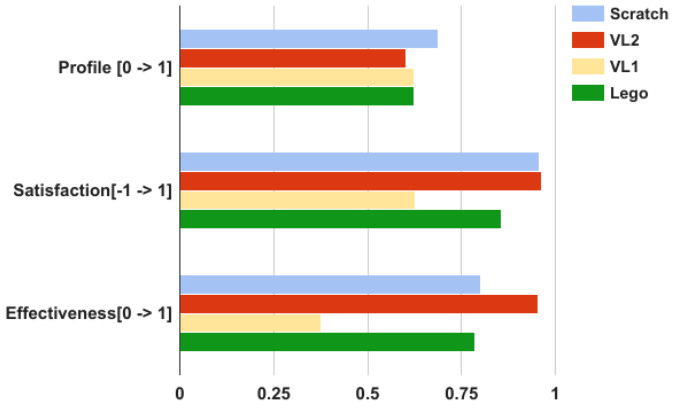


Figure 9.8: Experiment Results

tells how interesting and enjoyable they found the challenge itself; and, *Learnability* factor which tells how useful they found what was taught during the learning session which helped them to face the final challenge.

After the second experimental evaluation, we used a Games-Howell post-hoc test to determine which languages were significantly different from which languages, according to our set of characteristics under scrutiny. Figure 9.8 summarises this test's result, for the comparisons involving either VL1 or VL2, or both. We observe that VL1 lead to a significantly lower *Satisfaction* and *Effectiveness* when compared to Scratch, VL2 and Lego. VL1 was also significantly harder to learn than Scratch and VL2, but not significantly different when compared to Lego. In contrast, VL2 was consistently as good as Lego and Scratch, while superior to VL1 in terms of Satisfaction, Learnability and Effectiveness. We also manage to show that the usability of the Visualino (V2) significantly improved in terms of efficiency and satisfaction when compared to earlier version of the Visualino language (V1).

9.4.3 Conclusion of the case study

We reported how we involved teenagers, the end users, in several iterations of the engineering process of a programming language for low-cost robots and performed usability studies using empirical experiments. The preliminary evaluations helped in timely detecting crucial usability problems of Visualino, as well as detecting the audience that can comprehend the given paradigm. The provided feedback helped in improving the language to support the intended user's needs. A fundamental difference in this work is that the language evaluation was conducted as several empirical experiments introduced early in the DSL development process. Application of iterative usability assessments helped us to observed the convergence of the visual language to the degree of usability (regarding satisfaction and effectiveness) achieved by existing mature and commercial languages. We manage to reuse the same experiment's designs, which lower the cost of

usability assessment in each cycle (e.g. time needed for specification, and result analysis). Finally, we managed to obtain almost automatically the precise comparison between different Visualino versions (V1 and V2).

9.5 DSE Merge

We evaluated the tool support developed within this project at Budapest University of Technology and Economics, named DSE Merge [15] (Annex V), which presents a novel search-based automated model merge [115]. Final report of this project can be downloaded from public repository [12].

9.5.1 DSE Merge purpose

The DSE Merge was built on top of tools for model comparison and it uses guided rule-based Design Space Exploration (DSE) for merging models. Rule-based DSE aims to search and identify various design candidates which can fill in certain structural and numeric constraints. End user can specify operators which identify operations based on which exploration will travers paths from initial models. Many existing model merge approaches detect conflicts statically in a preprocessed phase. On other hand, DSE performs conflict detection dynamically, during exploration time. Further, it presents to the domain experts multiple consistent resolutions of conflicts, by allowing incorporation of domain-specific knowledge into the merge process by additional constraints, goals and operations. This approach is expected to provide better solutions. The alternative, i.e. baseline support for the model merge problem that is suitable for experimental comparison was found to have two possibilities: Diff Merge[13] and EMF Compare[14].

9.5.2 DSE Merge usability evaluation

The experiment took place on 11th December at the Budapest University of Technology and Economics. The general experimental process is presented in Fig.9.9, starting by Learning session, during which the subjects filled the Background questionnaire. After this they continue by to solve the exercises during Task session, that was video recorded. Finally, during Feedback session participants filled final questionnaire rating tools that they have used. The Figure 9.9 except reflecting the flow of activities during the experiment, explicitly shows documents and treatments that were provided to participants, as well as the instruments that were used to collect the data.

During the pilot session, the cognitive effort for each task was estimated to be similar, the TLX scale [15] is decided to be used just once for each tool that is being evaluated, at

[12] goo.gl/Kq3R1G
[13] http://www.eclipse.org/diffmerge/ (accessed September 19, 2017)
[14] https://www.eclipse.org/emf/compare/ (accessed September 19, 2017)
[15] https://humansystems.arc.nasa.gov/groups/TLX/ (accessed September 19, 2017)

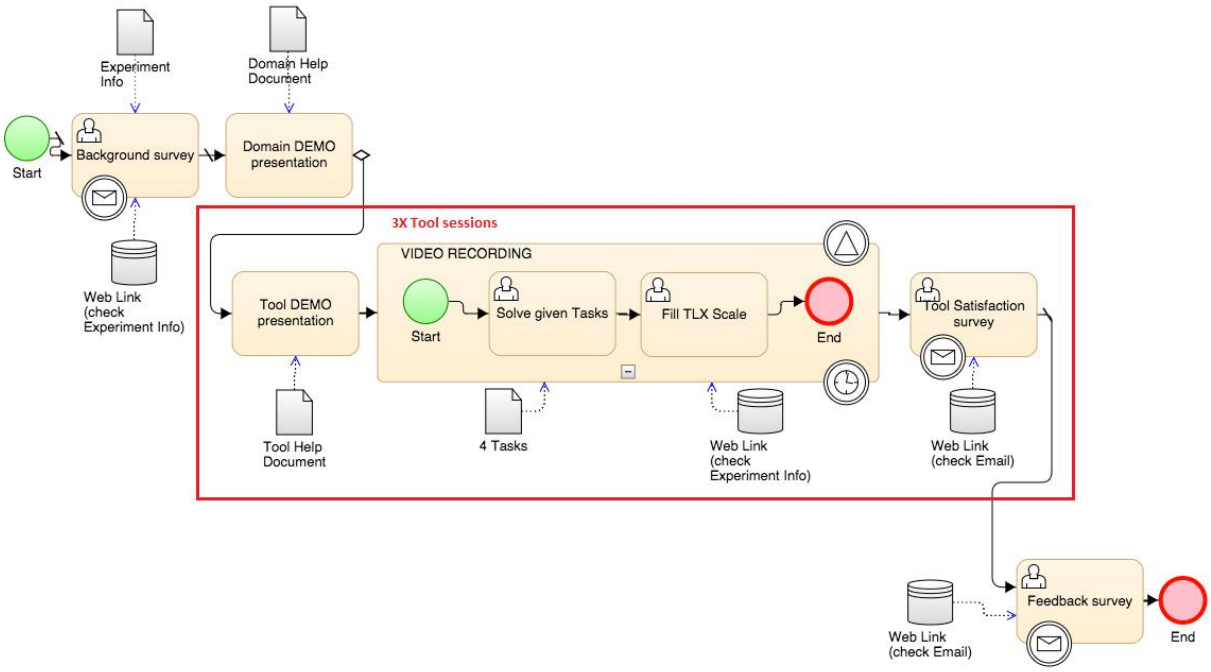


Figure 9.9: DSE Experiment Treatments (taken from [15])

the end of Video Session. Based on the results and opinions of the participants during the Pilot Session, it was found that Diff Merge is rated as a more competitive alternative for DSE Merge. The experimental groups were divided in two. One group started the Tool Session by learning about DiffMerge and than DSE Merge, while the second group had learned the other way around.

In regard to both groups, DSE Merge scored with lower time indicating a slightly better efficiency. Also, DSE Merge indicated slightly higher success rate (0.92 for DSE Merge, and 0.85 for Diff Merge) and explicit preference by 11/15 participants.

Concerning cognitive effort (see Figure 9.10), in total subjects rated with higher workload for Diff Merge regarding all factors, observing significantly higher Mental Demand and Frustration in comparison to which they experienced with DSE Merge.

Finally, related the Satisfaction, DSE Merge scored very high regarding easiness of use, expressiveness and learnability. Confidence was positive and better than with Diff Merge, while suitability to solve the given tasks even rated negatively for Diff Merge. User Interface, namely its readability and understandability, seems to be most important factor to be improved in order to provide better usability of the DSE Merge.

9.5.2.1 Conclusion of case study

Main contribution of this case study in a context of thesis was illustration of our experiment design proposed in Section 4.1 as a part of our research process (Figure 1.1). The case study contributes with the experiment design, instrumentation and metrics that can be easily repeated and reused for similar evaluations of new techniques. This experiment

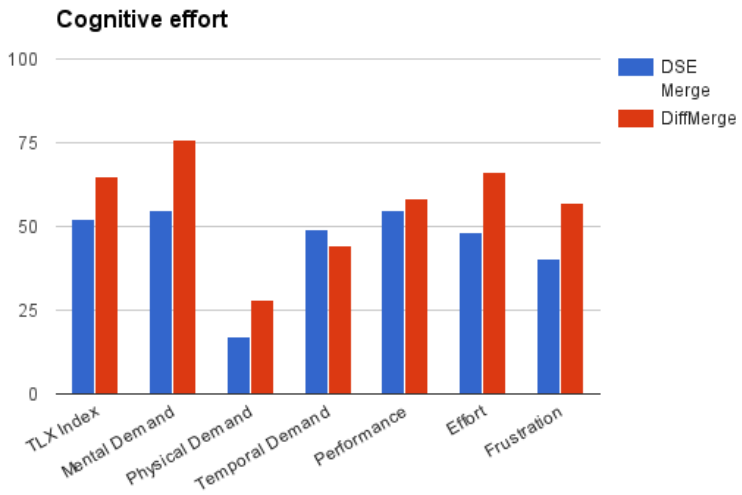


Figure 9.10: Cognitive Effort (taken from [15])

design took a deeper analysis of the subject's profiles, technology, social and physical environment and targeted workflow scenarios, which are defined explicitly and incorporated in a data collection instruments.

9.6 RDAL USE-ME requirement engineering approach

This case study, published in Software Language Engineering conference [25] (see Annex VI), was used as a part of USE-ME implementation validation of our research process (Figure 1.1). It served as illustration of integration of USE-ME conceptual framework (Chapter 5) with RDAL requirement approach [44]. This combination of existing languages and tools provides a comprehensive requirement engineering approach for DSL development and an interesting case study of languages composition allowing the reuse of the assets of the existing languages. The approach was illustrated with the development of the Visualino (Section 9.4, Annex IV).

9.6.1 Motivation for requirement engineering approach

As opposed to software and systems products for which several model-based Requirements Engineering approaches have been developed showing several benefits, DSLs nowadays is, to our knowledge, mostly developed informally without such support. A more formal and iterative approach is required to develop DSLs and track all requirements including the usability ones. As for other software products, the approach should include the context of use of the DSL in its environment, as well as the impact of recommendations with well-planned evaluation processes. Such an approach can be supported by modelling all these aspects using appropriate languages and tools.

The Requirements Definition and Analysis Language (RDAL) [43, 44] was developed as a fragmented language to be combined with other modelling languages in support of well-known requirement engineering best practices such as those recommended by the Requirements Engineering Management Handbook (REMH) [134] and those of GORE (Goal-Oriented Requirements Engineering) [197]. RDAL was originally planned to become a standard annexe of the Architecture Analysis and Design Language (AADL, SAE AS5506B standard) [16] for supporting requirements capture, analysis and verification. In the end, it led to the development of the ReqSpec language and ALISA (Architecture-Led Incremental System Assurance) approach [65], which are however strongly coupled with the AADL. In contrast, the modularity of RDAL allows its reuse with other Architecture Description Languages not necessarily targeting the safety-critical embedded systems domain. Although the REMH has been written for the domain of safety-critical embedded systems, a large majority of its practices that are supported by RDAL are generic enough to be applicable to the development of many other types of systems. As a matter of fact, the concepts of the RDAL-REMH approach share many similarities with the concepts required by USE-ME. The USE-ME focuses on usability and actually can benefit from a complete requirement engineering process on which it can base its evaluation.

9.6.2 Integration of RDAL and USE-ME

We first studied the concepts of both the RDAL-REMH and USE-ME approaches and compared them in order to evaluate the required effort and potential benefits of extending RDAL-REMH to support USE-ME. We showed that many RDAL-REMH concepts are also partially supported by USE-ME. However, in RDAL, there is no specific focus on usability and usability elements can only be identified by using the RDAL user-defined category system. Nevertheless, the modelling of all other requirement engineering concerns with RDAL can provide an essential basis for the USE-ME viewpoint on usability. Features provided by RDAL-REMH can be beneficial for USE-ME as it avoids redeveloping these constructs within USE-ME. A major issue that was found is the lack of language for representing the architecture design of a DSL in a similar fashion provided by AADL for embedded systems. Reusing AADL for modelling a DSL and its environment would not be appropriate due to the difference between the domains. This triggered the development of a language for the specification of DSL-based systems, namely the DSL-based Systems Specification Language (DSSL).

The purpose of the DSSL language is to model the design of a DSL being developed and the way it is used in its environment. Goals, requirements and environmental assumptions can then be assigned to elements of this representation of the DSL under development. Furthermore, DSSL can integrate descriptions of the syntaxes of the DSL implemented as an Ecore meta-model and potentially include graphical and/or textual concrete syntax(es) respectively represented as Sirius or/and Xtext grammar models.

[16] http://standards.sae.org/as5506b/ accessed September 19, 2017

The RDAL-REMH approach lacks the support of context-aware goal evaluation, which is necessary for usability evaluation. Usability evaluation is performed in a concrete context of use, e.g. with particular users, in a specific environment and performing selected scenarios. This means that after the validation of a usability goal and associated requirements, the results reflect only a partial scope. This is because it would be too expensive to perform evaluations taking all different combinations of the context model instances (e.g. using all possible robot configurations, testing all possible scenarios with participants, and having a significant number of participants having all possible combination of demographic and knowledge characteristics). Therefore, the USE-ME conceptual framework suggests a calculation of a *Success Coverage* of the usability goal, which will reflect the percentage of the scope which was taken into consideration during a validation, when compared to the complete context specification of the usability goal.

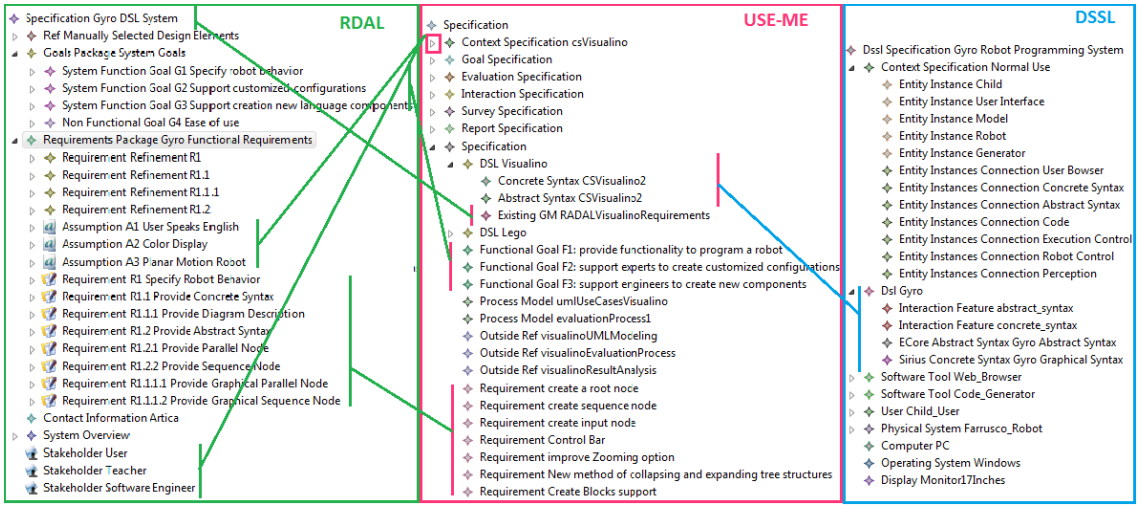


Figure 9.11: Integration points between RDAL, DSSL and USE-ME (taken from [25])

By integrating two approaches we enable usability evaluation to reuse and refer to the RDAL artefacts while performing context and goal modelling activities suggested by the USE-ME conceptual framework. On another hand, while applying the USE-ME activities it is likely that context and goal elements will be extended, or new ones discovered therefore directly contributing to the requirements refinement and DSL artefacts specification. We highlight interaction points and information flow between the RDAL-REMH and USE-ME approaches (see figure 9.11). The idea was to connect the RDAL non-functional goals referring to usability with a 'Quality in Use' root goal of the USE-ME Goal Model. This root goal represents the highest objective of the USE-ME conceptual framework (usability for all possible workflows, environment combinations and profiles).

9.6.3 Conclusion of case study

Our case study presented a usability-driven requirements engineering approach for DSL development. The RDAL-REMH approach used for embedded systems development with

the AADL language for system architectures has been adapted to DSL development by replacing the AADL with the DSSL language, a new DSL that we developed for modelling the DSL under development and its environment. We then provided a mapping between the USE-ME language and the combined RDAL-REMH languages for integrating the USE-ME usability driven development into the RDAL-REMH general requirement engineering approach.

Chapter 10

Conclusions

10.1 Thesis summary

This thesis had the main goal of presenting a solution for evaluating the usability of DSLs. We started by introducing our research objectives, formulating research questions and presenting our research approach (Chapter 1). We continued by understanding the relevant DSL context (its implementation, stakeholders and life-cycle) and the traditional procedure of usability evaluation of software products (Chapter 2). Further, we introduced a related work, highlighting a state of practice in DSL evaluation and identifying a usability evaluation approaches for GPLs and efforts which may lead to DSL usability evaluation, like the cases of application of UCD methodology and inclusion of the domain experts feedback into DSL development (Chapter 3).

We proposed to approach our problem by introducing an experiment design model which is general enough to be applied to different usability assessments. This model was further placed in a process of iterative UCD DSL development which was defined as a pattern language (Chapter 4). We captured all the relevant concepts and activities, by using a MDD approach, in a USE-ME conceptual framework (Chapter 5) which was evaluated internally by the NOVA-LINCS researchers knowledgeable about the DSL, MDD or usability.

From the point of view of feasibility of USE-ME conceptual framework to support the DSL usability evaluation, we have developed a declarative tool support integrable into DSL development infrastructure. The prototype was used to instantiate the usability evaluation of industrial DSL and was evaluated with the master students involved in DSL development projects (Chapter 6). Further, we illustrated how the USE-ME can be integrated with DSL development and discussed the applicability of USE-ME approach to incremental iterative DSL development, as well outside of the scope of MDD DSL

approach (Chapter 7). Finally, we discussed the potential users of the USE-ME approach.

The final step was to prove that our approach provides a solution to our research problems. In Chapter 8 we report on the experimental evaluation of the persons experienced in the DSL development or usability evaluation domain. We presented USE-ME evaluation following our evaluation methodology, i.e. by explicitly presenting the considered context. The evaluation corroborates our hypothesis.

We reported in Section 1.4, how we addressed our engineering problem through progressive evaluation with several case studies, on which we report details in Chapter 9. We reported our experience during the development of a RPG DSL following the regulative development cycle (Annex II) and our initial evaluation of the DSL named Pheasant (Annex I) as a part of our problem investigation. Further, two industrial case studies, FlowSL (Annex III) and Visualino (Annex IV), and one academic study, DSE Merge (Annex V), served to illustrate our solution design. Further, we combined approach with an existing requirement engineering approach for DSLs to show it's *integrability* with existing DSL development support (Annex VI).

10.2 Results obtained

The problem tackled in this thesis is very well-known in the area. To our knowledge before this thesis was written there was no real attempt to tackle the problem in such a global and methodical manner. Therefore, during our thesis argumentation, we believe to have introduced the systematic approach which promotes quality in use of DSLs, during their development process by leveraging usability as a first-class concern. We present a summary of the main results achieved and some of the benefits of the development of this dissertation:

1. To **address the problem of the absence of the systematic approach for the DSL usability evaluation we proposed a conceptual modelling framework called USE-ME**. This comprehensive framework, which is presented in Chapter 5, identifies all the mandatory concepts and activities and aggregates them into a formal metamodel. It highlights the complexity of the information that should be traced to streamline and automate the UCD process. The conceptual framework contributed directly to the thesis objective, by providing a set of practices that should be introduced to provide a complete solution to a complex problem of placing intended users as a focal point of DSLs design and conception. The framework helps the language engineers to explicitly model the evaluation process, which contributes to monitoring the impact of language evolution to the efficiency and effectiveness of practitioners using the language. We applied our approach in real-life case studies of the usability evaluation of a several DSLs.

2. To **promote usability concerns since an early stage of development of DSLs, we proposed a set of patterns and specified an iterative process which integrates**

well with the DSL development phases. The incremental nature of a typical DSL life cycle may also give the erroneous feeling that the language is being implicitly validated due to the intense interaction with the domain experts. The problem there is that the domain experts involved in the language development may not actually be the end users, and may therefore introduce biases in the perception of the language design and its usability. We proposed systematic approach which describe the best practices for application of usability evaluation methodologies during a DSL development process [21] (Section 4.2). The USE-ME conceptual framework supports specifying experimental assessments and tracing the impact of usability improvements since an early stage of development of DSLs. Further, the systematic approach, where usability concerns were promoted early, was applied in the context of the iterative development of a two industrial DSLs: DSL for humanitarian campaigns flow specification (FlowSL) [23] (Section 9.3); and a language which supports children to program a robot (Visualino) (Section 9.4).

3. To **integrate existing IDE support for development of DSLs with high Quality in Use, we developed a tool support** [27]. Despite the importance of patterns and comprehensive conceptual framework, they may not be sufficient to enable the language engineers to learn and use the systematic approach, in practice. The previously mentioned contributions would not be easily deployed if not supported by a modelling tool integrable into DSL development infrastructure. The USE-ME prototype [27] (Chapter 6) was integrated with an existing Eclipse-based IDE to support the development of DSLs. This prototype enables language engineers to take the role of Expert Evaluators and to prepare the evaluation models. Also, it supports tracing the evaluation goals, reusing experiment designs and providing better documentation and reasoning.

Some of the benefits arising from these results and confirmed by case studies:

- **Feasibility of the conceptual framework to support the language engineers to prepare evaluation models.** We found that evaluation experts knowledge can be transmitted to a typical DSL Engineer through the USE-ME conceptual framework. The framework was first validated by researchers from NOVA-LINCS research centre which were not involved in the framework development. These experts provided valuable feedback and improvement suggestions over the framework. However, people with high level of experience might also not be available during the DSL development process to take the role of Expert Evaluators. Therefore, we performed a preliminary pilot evaluation of the framework prototype with inexperienced engineers (i.e. master students in informatics) [28]. They found it easy to create test model specifications for the evaluation and a supporting tool expressive enough for the purpose of specifying usability evaluation. Finally, we conducted evaluation study with experts in community, which confirmed the feasibility of USE-ME

conceptual framework to support language engineers in conducting usability evaluation of DSLs.

- **Traceability of evaluation goals.** Our systematic approach enabled tracing a justification of the assessment, and its impact throughout the evolution of the DSL. Each evaluation goal is a concrete instantiation of a high-level usability objective. It refers to the precise context which stores the assumptions which impacted the evaluation decision. As a consequence of the evaluation, new features get discovered, and development priorities can change. The result of the previously performed evaluations alters its scope, as context awareness of the DSL stakeholders grows and changes over time. The evaluation impact gets smaller as the context knowledge gets wider.

- **Easy integration.** The given systematic approach does not depend on any particular technology. It was specified taking into consideration reuse of the existing knowledge defined in the common DSL development. Explicit integration points are designated as a part of a conceptual framework enabling easy integration with existing DSL artefacts or assessment support.

- **Reusable experiment design.** The evaluation modelling defines the experiment objects which are reusable. We showed in the case of Visualino (Section 9.4) that we managed to reuse the majority of the experiment design between iterations and benefit from this reuse by having a direct comparison between the current and previous version of the language.

10.3 Future work

In this section, we are pointing out the future work which emerged from the proposed methodological solution. Namely, we highlight the following two work directions to be explored:

Evolution of the tool. The USE-ME approach, besides documenting the evaluation process, enables reasoning about the models and setting a clear workflow to its users. As part of the future work, we are improving the tool support to guide the users through the modelling process and validate the models. It is being implemented as a part of Goal Coverage engine, which will contain specification and verification of different syntactic and semantic rules depending in which step of the USE-ME process is a user. This engine will also support the verification of the rules taken into account by the models in the different development stages. Finally, the USE-ME engine will be extended with a tool for (semi)automatically determine the success coverage of the specified usability goals by the DSL developer.

Identification and illustration of the integration points. One example was to join our approach with existing requirement based approaches for the development of DSLs.

Another perspective is the integration of support for experimental design, as well as a survey and interaction modelling support with the tool. Finally, another follow-up to the presented work is to study the integration of USE-ME with already existing solutions and modelling workbenches for the development of the DSLs. We believe that this contribution is a step towards the automation of the DSL development process tackling a specific concern that is typically neglected, Quality in use.

We highlight here issues which were not addressed in the context of this thesis:

Validation of iterative life cycle. The proposed USE-ME conceptual framework is claimed to support an iterative DSL development life cycle. In a real DSL development environment, it is hard and not realistic to apply two different development processes. In this case, one process should be implemented with the USE-ME conceptual framework and another without it to real case development of DSL. For this it is necessary to have organizations available to support investment into this two different projects and to support it with resources.

Reusability of the evaluation models in various context (reusing existing designs for the other DSLs). We pointed out the reusability of the produced USE-ME models and showed that it is possible to apply in the development of the same product in our two industrial case studies (Sections 9.3 and 9.4). However, we find that reusing and reasoning about different designs can be reused in different contexts. To make this possible, it is necessary to obtain an appropriate number of developed models, and organise them with the support of an Usability Catalogue. We leave this for future work as it is not the focus of this thesis.

Bibliography

[1] S. Aghaee and C. Pautasso. "End-User Development of Mashups with NaturalMash." In: *Journal of Visual Languages & Computing* 25.4 (2014), pp. 414–432. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2013.12.004.

[2] D. Albuquerque, B. Cafeo, A. Garcia, S. Barbosa, S. Abrahão, and A. Ribeiro. "Quantifying usability of domain-specific languages: An empirical study on software maintenance." In: *Journal of Systems and Software* 101 (2015), pp. 245–259. ISSN: 01641212. DOI: 10.1016/j.jss.2014.11.051.

[3] M. Alférez, U. Kulesza, A. Sousa, J. P. Santos, A. Moreira, J. Araújo, and V. Amaral. "A Model-driven Approach for Software Product Lines Requirements Engineering." In: *SEKE*. 2008, pp. 779–784.

[4] T. Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD-Books on Demand, 2010.

[5] R. S. Alroobaea, A. H. Al-Badi, and P. J. Mayhew. "Generating a Domain Specific Inspection Evaluation Method through an Adaptive Framework: A Comparative Study on Educational Websites." In: *International Journal of Human Computer Interaction (IJHCI)* 4.2 (2013), p. 88.

[6] M. Alva, A. Martínez P, J. Cueva L, T Sagástegui Ch, and B. López P. "Comparison of Methods and Existing Tools for the Measurement of Usability in the Web." In: *Web Engineering* (2003), pp. 386–389.

[7] V. Amaral. "Increasing productivity in High Energy Physics data mining with a Domain Specific Visual Query Language." In: *Phd. Thesis, University of Mannheim*. 2005.

[8] D. Amyot, A. Shamsaei, J. Kealey, E. Tremblay, A. Miga, G. Mussbacher, M. Alhaj, R. Tawhid, E. Braun, and N. Cartwright. "Towards advanced goal model analysis with jUCMNav." In: *International Conference on Conceptual Modeling*. Springer. 2012, pp. 201–210.

[9] M. Angelini, N. Ferro, G. Santucci, and G. Silvello. "VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis." In: *Journal of Visual Languages & Computing* 25.4 (2014), pp. 394–413. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2013.12.003.

[10] C. Ardito, M. F. Costabile, G. Desolda, R. Lanzilotti, M. Matera, A. Piccinno, and M. Picozzi. "User-driven visual composition of service-based interactive spaces." In: *Journal of Visual Languages & Computing* 25.4 (2014), pp. 278–296. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2014.01.003.

[11] C. Atkinson and T. Kühne. "Model Driven Development: A Metamodeling Foundation." In: *IEEE software* 20.5 (Sept. 2003), pp. 36–41. ISSN: 0740-7459. DOI: 10.1109/MS.2003.1231149.

[12] E. Azadi Marand, E. Azadi Marand, and M. Challenger. "DSML4CP: A Domain-specific Modeling Language for Concurrent Programming." In: *Computer Languages, Systems & Structures*. Vol. 44. 2015, pp. 319–341. DOI: 10.1016/j.cl.2015.09.002.

[13] A. Barišić. "Evaluating the Quality in Use of Domain-Specific Languages in an Agile Way." In: *Doctoral Symposium at the 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS), CEUR-WS* VOL-1071 (Sept. 2013). DOI: http://doi.org/10.5281/zenodo.889790.

[14] A. Barišić. "Iterative Evaluation of Domain-Specific Languages." In: *ACM Student Research Competition at the 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS)* 1115 (2013), pp. 100–105. DOI: http://doi.org/10.5281/zenodo.889889.

[15] A. Barišić. "STSM Report: Evaluating the efficiency in use of search-based automated model merge technique." In: *Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)*. COST action IC1404. European cooperation in science and technology (COST). European cooperation in science and technology (COST), 2016. DOI: 10.5281/ZENODO.237420.

[16] A. Barišić. "Framework support for Usability evaluation of Domain-Specific Languages." In: *In ACM Student Research Competition (SRC) at the Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)*. ACM. Vancouver, British Columbia, Canada: ACM, 2017. DOI: https://doi.org/10.1145/3135932.3135953.

[17] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "How to reach a usable dsl? Moving toward a systematic evaluation." In: *Electronic Communications of the EASST: 5th Int. Workshop on Multi-paradigm Modeling (MPM 2011)* 50 (2011), p. 13. DOI: 10.14279/tuj.eceasst.50.741.

[18] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "Quality in use of domain-specific languages: a case study." In: *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU) at SPLASH*. PLATEAU '11 (2011), pp. 65–72. DOI: 10.1145/2089155.2089170.

[19] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "Quality in Use of DSLs: Current Evaluation Methods." In: *Proceedings of the 3rd INForum - Simpósio de Informática (INForum2011)* (Sept. 2011). DOI: http://doi.org/10.5281/zenodo.889940.

[20] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "Evaluating the Usability of Domain-Specific Languages." In: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments* (2012). Ed. by M. Mernik, pp. 386–407. DOI: 10.4018/978-1-4666-2092-6.

[21] A. Barišić, P. Monteiro, V. Amaral, M. Goulão, and M. P. Monteiro. "Patterns for Evaluating Usability of Domain-Specific Languages." In: *Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012*. Tucson, Arizona: The Hillside Group, Oct. 2012, 14:1–14:34. DOI: http://doi.org/10.5281/zenodo.889927.

[22] A. Barišić, V. Amaral, and M. Goulão. "Usability evaluation of domain-specific languages." In: *SEDES Doctoral Symposium at the 8th International Conference on the Quality of Information and Communications Technology (QUATIC)* (2012), pp. 342–347. DOI: 10.1109/QUATIC.2012.63.

[23] A. Barišić, V. Amaral, M. Goulão, and A. Aguiar. "Introducing Usability Concerns Early in the DSL Development Cycle : FlowSL Experience Report." In: *MD2P2 2014 Model-Driven Development Processes and Practices Workshop Proceedings, CEUR-WP* VOL-1249 (2014). DOI: http://doi.org/10.5281/zenodo.889685.

[24] A. Barišić, M. Goulão, and V. Amaral. "Domain-Specific Language domain analysis and evaluation: a systematic literature review." In: *Faculdade de Ciencias e Technologia, Universidade Nova da Lisboa* (2015). DOI: 10.5281/ZENODO.265487.

[25] A. Barišić, D. Blouin, V. Amaral, and M. Goulão. "A Requirements Engineering Approach for Usability-Driven DSL Development." In: *10th International Conference on Software Language Engineering (SLE)*. Vancouver, British Columbia, Canada: ACM, 2017. DOI: 10.1145/3136014.3136027.

[26] A. Barišić, V. Amaral, and M. Goulão. "Usability Driven DSL development with USE-ME." In: *Computer Languages, Systems and Structures (ComLan)* ISBN 1477- (2017). DOI: https://doi.org/10.1016/j.cl.2017.06.005.

[27] A. Barišić, V. Amaral, and M. Goulão. "Usability Software Engineering - Modeling Environment (USE-ME 1.1)." In: *Faculdade de Ciências e Tecnologia, Universidade Nova da Lisboa* (2017). DOI: 10.5281/ZENODO.345941.

[28] A. Barišić, V. Amaral, and M. Goulão. *USE-ME Empirical evaluation pilot study*. Faculdade de Ciencias e Technologia, Universidade Nova da Lisboa, 2017. DOI: 10.5281/ZENODO.398830.

[29] A. Barišić, J. Cambeiro, V. Amaral, M. Goulão, and T. Mota. "Leveraging Teenagers Feedback in the Development of a Domain-Specific Language." In: *The 33rd ACM/SIGAPP Symposium On Applied Computing*. Pau, France: ACM, 2018. DOI: doi.org/10.1145/3167132.3167264.

[30] B. Barroca and V. Amaral. "(H)ALL: a DSVL for designing user interfaces for Control Systems." In: *Proceedings of the 5th Nordic Workshop on Model Driven Engineering NW-MoDE 2007 27-29 August 2007*. Blekinge Institute of Technology, 2007.

[31] V. R. Basili. "The role of experimentation in software engineering: past, current, and future." In: *18th International Conference on Software Engineering (ICSE'1996)*. Berlin, Germany: IEEE Computer Society, 1996, pp. 442–449.

[32] V. R. Basili. "The Role of Controlled Experiments in Software Engineering Research." In: *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Ed. by V. R. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, and R. Selby. LNCS. Springer Berlin / Heidelberg, 2007, pp. 33–37.

[33] E. Bauleo, S. Carnevale, T. Catarci, S. Kimani, M. Leva, and M. Mecella. "Design, realization and user evaluation of the SmartVortex Visual Query System for accessing data streams in industrial engineering applications." In: *Journal of Visual Languages & Computing* 25.5 (2014), pp. 577–601. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2014.08.002.

[34] K. Beck, M. Fowler, and G. Beck. "Bad smells in code." In: *Refactoring: Improving the design of existing code* (1999), pp. 75–88.

[35] R. Bellamy, B. John, J. Richards, and J. Thomas. "Using CogTool to model programming tasks." In: *Evaluation and Usability of Programming Languages and Tools (PLATEAU 2010)* (2010), p. 1.

[36] D. Benyon, T. Green, and D. Bental. *Conceptual modeling for user interface development*. Springer Science & Business Media, 2012. ISBN: 978-1-85233-009-5. DOI: 10.1007/978-1-4471-0797-2.

[37] N. Bevan. "Quality and usability: a new framework." In: *Achieving software product quality, van Veenendaal, E, and McMullan, J (eds) Tutein Nolthenius, Netherlands* (1997).

[38] N. Bevan. "Cost benefits framework and case studies." In: *Cost-Justifying Usability: An Update for the Internet Age. Morgan Kaufmann* (2005).

[39] R. G. Bias and D. J. Mayhew. *Cost-justifying usability: An update for the Internet age*. Elsevier, 2005.

[40] L. Binucci. "Software quality of use: Evaluation by MUSiC." In: *Objective Software Quality* (1995), pp. 165–178.

[41] A. Blackwell and T. Green. "Notational systems-the cognitive dimensions of notations framework." In: *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann* (2003).

[42] A. F. Blackwell. "Palimpsest: A layered language for exploratory image processing." In: *Journal of Visual Languages & Computing* 25.5 (2014), pp. 545–571. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2014.07.001.

[43] D. Blouin and H. Giese. "Combining Requirements, Use Case Maps and AADL Models for Safety-Critical Systems Design." In: *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016, pp. 266–274. ISBN: 978-1-5090-2820-7.

[44] D. Blouin, E. Senn, and S. Turki. "Defining an annex language to the architecture analysis and design language for requirements engineering activities support." In: *2011 Model-Driven Requirements Engineering Workshop*. IEEE, 2011, pp. 11–20. ISBN: 978-1-4577-0957-9.

[45] A. Brooks. "Meta Analysis -A Silver Bullet - for Meta-Analysts." In: *Empirical Software Engineering* 2.4 (1997), pp. 333–338.

[46] F. Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975. ISBN: 0-201-00650-2.

[47] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. "A system of patterns: Pattern-oriented software architecture." In: *Wiley* (1996).

[48] F. Campagne. *The MPS Language Workbench, Volume I*. CreateSpace Independent Publishing Platform, 2014.

[49] J. L. Cánovas Izquierdo and J. Cabot. "Enabling the collaborative definition of DSMLs." In: *Advanced Information Systems Engineering*. Springer. 2013, pp. 272–287.

[50] J. L. Cánovas Izquierdo and J. Cabot. "Collaboro: a collaborative (meta) modeling tool." In: *PeerJ Computer Science* 2 (Oct. 2016), e84. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.84.

[51] J. L. Cánovas Izquierdo, J. Cabot, J. J. López-Fernández, J. S. Cuadrado, E. Guerra, and J. de Lara. "Engaging end-users in the collaborative development of domain-specific modelling languages." In: *Cooperative Design, Visualization, and Engineering*. Springer, 2013, pp. 101–110.

[52] P. Carlshamre. "A usability perspective on requirement engineering." In: *From methodology to product development, Doctorial Thesis* 726 (2001).

[53] T Catarci. "What happened when database researchers met usability." In: *Information Systems* 25.3 (2000), pp. 177–212. ISSN: 0306-4379.

[54] I. Čeh, M. Črepinšek, T. Kosar, and M. Mernik. "Ontology driven development of domain-specific languages." In: *Computer Science and Information Systems* 8.2 (2011), pp. 317–342. ISSN: 1820-0214. DOI: 10.2298/CSIS101231019C.

[55] M. Challenger. "A Systematic Approach on Evaluating Domain-specific Modeling Languages for Multiagent Systems." In: *Software Quality Journal* 24.3 (2016), pp. 755–795.

[56] M. Challenger, S. Demirkol, S. Getir, M. Mernik, G. Kardas, and T. Kosar. "On the use of a domain-specific modeling language in the development of multiagent systems." In: *Engineering Applications of Artificial Intelligence* 28 (Feb. 2014), pp. 111–141. ISSN: 0952-1976. DOI: 10.1016/J.ENGAPPAI.2013.11.012.

[57] H. Cho, J. Gray, and E. Syriani. "Creating visual domain-specific modeling languages from end-user demonstration." In: *Proceedings of the 4th International Workshop on Modeling in Software Engineering*. IEEE Press. 2012, pp. 22–28.

[58] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media, 2012.

[59] A. Cockburn and J. Highsmith. "Agile software development: The people factor." In: *Computer* 34.11 (2001), pp. 131–133.

[60] S. Cook, G. Jones, S. Kent, and A. C. Wils. *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley Professional, 2007.

[61] D. Correal and R. Casallas. "Using Domain Specific Languages for Software Process Modeling." In: *OOPSLA Workshop on Domain-Specific Modeling*. Montreal, Canada, 2007.

[62] F. Cuenca, J. V. d. Bergh, K. Luyten, and K. Coninx. "A user study for comparing the programming efficiency of modifying executable multimodal interaction descriptions: a domain-specific language versus equivalent event-callback code." In: *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU 2015*. New York, New York, USA: ACM Press, 2015, pp. 31–38. ISBN: 9781450339070. DOI: 10.1145/2846680.2846686.

[63] K. Czarnecki. "Overview of generative software development." In: *Unconventional Programming Paradigms*. Springer, 2005, pp. 326–341.

[64] J. Danado and F. Paternò. "Puzzle: A mobile application development environment using a jigsaw metaphor." In: *Journal of Visual Languages & Computing* 25.4 (2014), pp. 297–315. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2014.03.005.

[65] J. Delange, P. Feiler, and E. Neil. "Incremental Life Cycle Assurance of Safety-Critical Systems." In: (Jan. 2016).

[66] A. Dix. *Human computer interaction*. Pearson Education, 2004.

[67] J. S. Dumas and J. C. Redish. "A Practical Guide to Usability Testing, intellectTM." In: *Exeter, England* (1999).

[68] S. Erdweg, T. van der Storm, M. Völter, L. Tratt, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, K. van der Vlist, G. Wachsmuth, and J. van der Woning. "Evaluating and comparing language workbenches: Existing results and benchmarks for the future." In: *Computer Languages, Systems & Structures* 44 (2015), pp. 24–47. ISSN: 14778424. DOI: 10.1016/j.cl.2015.08.007.

[69] A. Ewais and O. De Troyer. "A Usability Evaluation of Graphical Modelling Languages for Authoring Adaptive 3D Virtual Learning Environments." In: *CSEDU (1)*. 2014, pp. 459–466.

[70] A. Fernandez, E. Insfran, and S. Abrahão. "Usability evaluation methods for the web: A systematic mapping study." In: *Information and Software Technology* 53.8 (Aug. 2011), pp. 789–817. ISSN: 09505849. DOI: 10.1016/j.infsof.2011.02.007.

[71] D. Fogli and L. P. Provenza. "A meta-design approach to the development of e-government services." In: *Journal of Visual Languages & Computing* 23.2 (2012), pp. 47–62. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2011.11.003.

[72] E. Folmer and J. Bosch. "Usability patterns in software architecture." In: *Proceedings of the 10th International Conference on Human-Computer Interaction (HCI 2003)*. 2003, pp. 93–97.

[73] M. Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004. ISBN: 0321193687.

[74] M. Fowler. *Domain Specific Languages*. 1st. Addison-Wesley Professional, 2010. ISBN: 0321712943, 9780321712943.

[75] J Freeze. "Creating DSLs with Ruby." In: *Ruby Code & Style* 16 (2006).

[76] P. Gabriel, M. Goulão, and V. Amaral. "Do Software Languages Engineers Evaluate their Languages?" In: *XIII Congreso Iberoamericano en "Software Engineering"(CIbSE'2010), ISBN: 978-9978-325-10-0*. Ed. by X. Franch, I. Gimenes, and J.-P. Carvallo. Cuenca, Ecuador: Universidad del Azuay, 2010, pp. 149–162.

[77] A. García Frey, E. Céret, S. Dupuy-Chessa, and G. Calvary. "QUIMERA." In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '11*. New York, New York, USA: ACM Press, 2011, p. 265. ISBN: 9781450306706. DOI: 10.1145/1996461.1996534.

[78] I. Gatto and F. Pittarello. "Creating Web3D educational stories from crowdsourced annotations." In: *Journal of Visual Languages & Computing* 0 (2014), pp. –. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2014.10.010.

[79] M Gellner and P Forbrig. "A Usability Evaluation Pattern Language." In: *9nd INTERACT*. Zurich, Switzerland, 2003.

[80] I. Gibbs, S. Dascalu, and F. Harris. "A separation-based UI architecture with a DSL for role specialization." In: *Journal of Systems and Software* 101 (Mar. 2015), pp. 69–85. ISSN: 0164-1212. DOI: 10.1016/J.JSS.2014.11.039.

[81] M. Giordano, G. Polese, G. Scanniello, and G. Tortora. "A system for visual role-based policy modelling." In: *Journal of Visual Languages & Computing* 21.1 (2010), pp. 41–64. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2009.11.002.

[82] P. Giorgini, J. Mylopoulos, and R. Sebastiani. "Goal-oriented requirements analysis and reasoning in the tropos methodology." In: *Engineering Applications of Artificial Intelligence* 18.2 (2005), pp. 159–171.

[83] M. Goulão. "Component-Based Software Engineering: a Quantitative Approach." Doctoral dissertation. 2008.

[84] M. Goulão and F. B. Abreu. "Modeling the Experimental Software Engineering Process." In: *6th International Conference on the Quality of Information and Communications Technology (QUATIC'2007)*. Ed. by R. Machado. Lisbon, Portugal: IEEE Computer Society, 2007, pp. 77–90.

[85] I. Graham. *A pattern language for Web usability*. Addison-Wesley, 2003, p. 283. ISBN: 0201788888.

[86] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle. "Domain-specific modeling." In: *Handbook of Dynamic System Modeling* (2007), pp. 1–7.

[87] J. Gray, K. Fisher, C. Consel, G. Karsai, M. Mernik, and J.-P. Tolvanen. "DSLs: the good, the bad, and the ugly." In: *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. ACM. 2008, pp. 791–794.

[88] T. R. G. Green and M. Petre. "Usability analysis of visual programming environments: a cognitive dimensions framework." In: *Journal of Visual Languages & Computing* 7.2 (1996), pp. 131–174.

[89] J. Greenfield and K. Short. "Software factories: assembling applications with patterns, models, frameworks and tools." In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM. 2003, pp. 16–27.

[90] R. C. Gronback. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

[91] J. C. Grundy, J. G. Hosking, R. W. Amor, W. B. Mugridge, and Y Li. "Domain-Specific visual languages for specifying and generating data mapping systems." In: *Journal of Visual Languages and Computing, Elsevier* 15 (2004), pp. 243–263.

[92] S. Günther. "Development of internal domain-specific languages." In: *Proceedings of the 18th Conference on Pattern Languages of Programs - PLoP '11*. New York, New York, USA: ACM Press, 2011, pp. 1–25. ISBN: 9781450312837. DOI: 10.1145/2578903.2579139.

[93] A. Haase, M. Völter, S. Efftinge, and B. Kolb. "Introduction to openArchitectureWare 4.1. 2." In: *MDD Tool Implementers Forum*. 2007.

[94] M. A. K. Halliday. *Language as social semiotic*. London Arnold, 1978.

[95] F. Häser, M. Felderer, and R. Breu. "An Integrated Tool Environment for Experimentation in Domain Specific Language Engineering." In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (2016). DOI: 10.1145/2915970.2916010.

[96] F. Häser, M. Felderer, and R. Breu. "Is business domain language support beneficial for creating test case specifications: A controlled experiment." In: *Information and Software Technology* 79 (2016), pp. 52–62. ISSN: 09505849. DOI: 10.1016/j.infsof.2016.07.001.

[97] Haugen and P. Mohagheghi. "A Multi-dimensional Framework for Characterizing Domain Specific Languages." In: *OOPSLA Workshop on Domain-Specific Modeling*. Montréal, Canada, 2007.

[98] F. Hermans, M. Pinzger, and A. V. Deursen. "Domain-Specific Languages in Practice: A User Study on the Success Factors." In: *12th International Conference on Model Driven Engineering Languages and Systems*. Vol. 5795/2009. Denver, Colorado, USA: Lecture Notes in Computer Science, 2009, pp. 423–437.

[99] P. Hudak. "Building Domain-specific Embedded Languages." In: *ACM Comput. Surv.* 28.4es (Dec. 1996). ISSN: 0360-0300. DOI: 10.1145/242224.242477.

[100] A. Hussey. "Patterns for safer human-computer interfaces." In: *SAFECOMP*. Vol. 99. Springer. 1999, pp. 103–112.

[101] International Organization for Standardization. *ISO/IEC 25022:2016, Systems and software engineering - Systems and software quality requirements and evaluation (SQuaRE) - Measurement of quality in use*.

[102] International Organization for Standardization. "ISO/IEC 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability." Doctoral dissertation. International Organization for Standardization (ISO), www.iso.org, June 2001.

[103] International Organization for Standardization. *ISO/IEC 9126: Information Technology - Software Product Evaluation - Software Quality Characteristics and Metrics*. Tech. rep. International Standard Organization, 2004.

[104] International Organization for Standardization. "ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models." Doctoral dissertation. International Organization for Standardization (ISO), www.iso.org, Mar. 2011.

[105] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. "Reporting Experiments in Software Engineering." In: *Guide to advanced empirical software engineering*. Ed. by F. Shull, J. Singer, and D. I. Sjøberg. Vol. 5971. London: Springer-Verlag, 2008.

[106] A. N. Johanson and W. Hasselbring. "Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment." In: *Empirical Software Engineering* (Nov. 2016), pp. 1–31. ISSN: 1382-3256. DOI: 10.1007/s10664-016-9483-z.

[107] J. Johnson and A. Henderson. "Conceptual models: begin by designing what to design." In: *interactions* 9.1 (2002), pp. 25–32. ISSN: 1072-5520.

[108] C. Jones. "Software change management." In: *Computer* 29.2 (1996), pp. 80–82. ISSN: 00189162. DOI: 10.1109/2.485858.

[109] M. Kabáč, N. Volanschi, and C. Consel. "An evaluation of the DiaSuite toolset by professional developers: learning cost and usability." In: *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU 2015*. New York, New York, USA: ACM Press, 2015, pp. 9–16. ISBN: 9781450339070. DOI: 10.1145/2846680.2846682.

[110] G. Kahraman and S. Bilgen. "A framework for qualitative assessment of domain-specific languages." In: *Software & Systems Modeling* 14.4 (2013), 1505–1526.

[111] J. Kärnä, J.-P. Tolvanen, and S. Kelly. "Evaluating the use of domain-specific modeling in practice." In: *Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling*. 2009.

[112] S. Kelly and R. Pohjonen. "Worst Practices for Domain-Specific Modeling." In: *IEEE Software* 26.4 (2009), pp. 22–29.

[113] S. Kelly and J.-P. Tolvanen. *Domain-specific modeling: enabling full code generation*. Wiley-Interscience, 2008, p. 427. ISBN: 0470249250.

[114] R. Kendall, J. C. Carver, D. Fisher, D. Henderson, A. Mark, D. Post, C. E. Rhoades Jr., and S. Squires. "Development of a Weather Forecasting Code: A Case Study." In: *IEEE Software*. Vol. 25. 4. July 2008, pp. 59–65. DOI: 10.1109/MS.2008.86.

[115] M. Kessentini, P. Langer, and M. Wimmer. "Searching models, modeling search: On the synergies of SBSE and MDE." In: *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*. IEEE Press. 2013, pp. 51–54.

[116] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton. "A software engineering experiment in software component generation." In: *Proceedings of the 18th international conference on Software engineering (ICSE'1996)*. IEEE Computer Society. 1996, p. 552. ISBN: 0818672463.

[117] B. A. Kitchenham and S. L. Pfleeger. "Personal opinion surveys." In: *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 63–92.

[118] A. G. Kleppe. *Software language engineering: creating domain-specific languages using metamodels*. Addison-Wesley, 2009. ISBN: 0321553454.

[119] A. Kobsa. "Generic user modeling systems." In: *User modeling and user-adapted interaction*. Vol. 11. 1-2. Springer, 2001, pp. 49–63.

[120] D. Kolovos, L. Rose, R. Paige, and A Garcia-Dominguez. "The epsilon book." In: *Structure* 178 (2010), p. 39.

[121] D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. C. Polack. "Requirements for domain-specific languages." In: *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*. Vol. 2006. 2006.

[122] T. Kosar, S. Bohra, and M. Mernik. "Domain-Specific Languages: A Systematic Mapping Study." In: *Information and Software Technology* 71 (2016), pp. 77–91.

[123] T. Kosar, P. E. M. López, P. A. Barrientos, and M. Mernik. "A preliminary study on various implementation approaches of domain-specific language." In: *Information and Software Technology* 50.5 (2008), pp. 390–405.

[124] T. Kosar, N. Oliveira, M. Mernik, M. J. V. Pereira, M. Crepinšek, D. Cruz, and P. R. Henriques. "Comparing General-Purpose and Domain-Specific Languages: An Empirical Study." In: *Computer Science and Information Systems* 7.2 (2010), pp. 247–264.

[125] T. Kosar, M. Mernik, and J. Carver. "Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments." In: *Empirical Software Engineering* 17.3 (2011), 276–304. DOI: 10.1109/MS.2003.1231149.

[126] T. Kühne. "Matters of (meta-) modeling." In: *Software & Systems Modeling* 5.4 (2006), pp. 369–385.

[127] M. Kuhrmann, G. Kalus, and A. Knapp. "Rapid Prototyping for Domain-specific Languages-From Stakeholder Analyses to Modelling Tools." In: *Enterprise Modelling and Information Systems Architectures* 8.1 (2013), pp. 62–74.

[128] C. Lakos. "From Coloured Petri Nets to Object Petri Nets." In: Springer, Berlin, Heidelberg, 1995, pp. 278–297. DOI: 10.1007/3-540-60029-9{_}45.

[129] M. Lárusdóttir, Cajander, and J. Gulliksen. "Informal feedback rather than performance measurements: user-centred evaluation in Scrum projects." In: *Behaviour & Information Technology* ahead-of-print (2013), pp. 1–18.

[130] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. "The generic modeling environment." In: *Workshop on Intelligent Signal Processing, Budapest, Hungary*. Vol. 17. 2001.

[131] N. L. Leech, K. C. Barrett, and G. A. Morgan. *SPSS for intermediate statistics: Use and interpretation*. Psychology Press, 2005. ISBN: 1-4106-1142-6.

[132] LEGO. *Lego Mindstorms*. 2015.

[133] D. L. Lempia and S Miller. *Requirements engineering management findings report*. Tech. rep. Technical report DOT/FAA/AR-08/34, Federal Aviation Administration, 2009.

[134] D. L. Lempia and S. P. Miller. "Requirements engineering management handbook." In: *National Technical Information Service (NTIS)* 1 (2009).

[135] P. Leonardo. "Child Programming : An adequate Domain Specific Language for programming specific robots." In: (2013).

[136] D. Leroux, M. Nally, and K. Hussey. "Rational Software Architect: A tool for domain-specific modeling." In: *IBM Systems Journal* 45.3 (2006), pp. 555–568.

[137] X.-W. T. Liu. "An intuitive and flexible architecture for intelligent mobile robots." Doctoral dissertation. The University of Manitoba, 2005.

[138] M. Mahemoff and L. Johnston. "Usability Pattern Languages: the"Language"Aspect." In: *INTERACT*. 2001, pp. 350–358.

[139] A. Marcus. "The ROI of Usability." In: *Cost-Justifying Usability*. Ed. by Bias and Mayhew. North- Holland: Elsevier, 2004.

[140] J. Marôco. *Análise estatística com o SPSS Statistics*. 2nd. Lisbon: ReportNumber, Lda, 2011.

[141] E. Marques, V. Balegas, B. Barroca, A. Barišić, and V. Amaral. "The RPG DSL: a case study of language engineering using MDD for Generating RPG Games for Mobile Phones." In: *Proceedings of the 12th workshop on Domain-specific modeling* (2012), pp. 13–18. DOI: 10.1145/2420918.2420923.

[142] R. C. Martin and P. Hall. *Agile Software Development Principles, Patterns, and Practices Alan Apt Series*. 2002.

[143] D. J. Mayhew. "The usability engineering lifecycle." In: *CHI'99 Extended Abstracts on Human Factors in Computing Systems*. ACM. 1999, pp. 147–148.

[144] D. McKean and J. Sprinkle. "Heterogeneous multi-core systems: UML profiles vs. DSM Approaches." In: *Proceedings of the 2012 workshop on Domain-specific modeling*. ACM. 2012, pp. 45–48.

[145] S. Meliá, C. Cachero, J. M. Hermida, and E. Aparicio. "Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study." In: *Software Quality Journal* 24.3 (Sept. 2016), pp. 709–735. ISSN: 0963-9314. DOI: 10.1007/s11219-015-9299-x.

[146] J. Merilinna and J. Perssinen. "Comparison Between Different Abstraction Level Programming: Experiment Definition and Initial Results." In: *OOPSLA Workshop on Domain-Specific Modeling*. Montreal, Canada, 2007.

[147] M. Mernik and V. Žumer. "Incremental programming language development." In: *Computer Languages, Systems & Structures* 31.1 (2005), pp. 1–16. ISSN: 14778424. DOI: 10.1016/j.cl.2004.02.001.

[148] M. Mernik, J. Heering, and A. M. Sloane. "When and How to Develop Domain-Specific Languages." In: *ACM Computing Surveys* 37.4 (2005), pp. 316–344.

[149] J. Miller. "Applying Meta-Analytical Procedures to Software Engineering Experiments." In: *Journal of Systems and Software* 54.11 (2000), pp. 29–39.

[150] M. P. Monteiro. "On the Cognitive Foundations of Modularity." In: *Psychology of Programming Interest Group Conference.* Academic Press. González-Morales, D., de Antonio, LMM, & García, JLR (2011). Teaching soft skills in software engineering. In Proceedings of Global Engineering Education Conference, 2011, pp. 630–637.

[151] D. Moody and J. van Hillegersberg. "Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams." In: *Software Language Engineering* (2009), pp. 16–34.

[152] N. S. Murray, N. W. Paton, C. A. Goble, and J Bryce. "Kaleidoquery: a flow-based visual language and its evaluation." In: *Journal of Visual Languages & Computing* 11.2 (2000), pp. 151–189. ISSN: 1045-926X.

[153] C. Neumann, R. A. Metoyer, and M. Burnett. "End-user strategy programming." In: *Journal of Visual Languages & Computing* 20.1 (2009), pp. 16–29. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2008.04.005.

[154] J. Nielsen and R. Molich. "Heuristic evaluation of user interfaces." In: *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people* (1990), pp. 249–256.

[155] J. Nielsen. *Usability Engineering*. Academic Press, 1993.

[156] H. Nishino. "Misfits in abstractions: towards user-centered design in domain-specific languages for end-user programming." In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM. 2011, pp. 215–216.

[157] D. A. Norman and S. W. Draper. "User centered system design." In: *Hillsdale, NJ* (1986).

[158] T. O'Reilly. "What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software." In: (2005).

[159] A. Öztaş and Ökmen. "Risk analysis in fixed-price design–build construction projects." In: *Building and Environment* 39.2 (Feb. 2004), pp. 229–237. ISSN: 03601323. DOI: 10.1016/j.buildenv.2003.08.018.

[160] J. F. Pane. "A programming system for children that is designed for usability." Doctoral dissertation. Lewis, University, 2002.

[161] F. Paz and J. A. Pow-Sang. "Current Trends in Usability Evaluation Methods: A Systematic Review." In: *2014 7th International Conference on Advanced Software Engineering and Its Applications*. IEEE, Dec. 2014, pp. 11–15. ISBN: 978-1-4799-7761-1. DOI: 10.1109/ASEA.2014.10.

[162] F. Paz and J. A. Pow-Sang. "A Systematic Mapping Review of Usability Evaluation Methods for Software Development Process." In: *International Journal of Software Engineering and Its Applications* 10.1 (2016), pp. 165–178. ISSN: 1738-9984. DOI: 10.14257/ijseia.2016.10.1.16.

[163] H. Petrie and N. Bevan. "The evaluation of accessibility, usability and user experience." In: *The Universal Access Handbook*. Ed. by C. Stephanidis. Human Factors and Ergonomics. CRC Press, 2009.

[164] I. Poltronieri Rodrigues, M. de Borba Campos, and A. F. Zorzo. "Usability Evaluation of Domain-Specific Languages: A Systematic Literature Review." In: Springer, Cham, July 2017, pp. 522–534. DOI: 10.1007/978-3-319-58071-5{_}39.

[165] L. Prechelt. "An Empirical Comparison of Seven Programming Languages." In: *IEEE Computer* 33.10 (2000), pp. 23–29.

[166] V. Rajlich and N. Wilde. "The role of concepts in program comprehension." In: *Program Comprehension, 2002. Proceedings. 10th International Workshop on*. IEEE. 2002, pp. 271–278.

[167] P. Reisner. "Human factors studies of database query languages: A survey and assessment." In: *ACM Computing Surveys (CSUR)* 13.1 (1981), pp. 13–31. ISSN: 0360-0300.

[168] P. Reisner. "Query languages." In: *Handbook of Human-Computer Interaction, North-Holland, Amsterdam, The Netherlands* (1988), pp. 257–280.

[169] C. Righi and J. J. S. James. *User-centered design stories : real-world UCD case files*. Elsevier/Morgan Kaufman, 2007, p. 535. ISBN: 9780123706089.

[170] J. Roth. "Patterns of Mobile Interaction." In: *Personal and Ubiquitous Computing* 6.4 (Sept. 2002), pp. 282–289. ISSN: 1617-4909. DOI: 10.1007/s007790200029.

[171] J. M. Rouly, J. D. Orbeck, and E. Syriani. "Usability and Suitability Survey of Features in Visual Ides for Non-Programmers." In: *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU '14*. New York, New York, USA: ACM Press, 2014, pp. 31–42. ISBN: 9781450322775. DOI: 10.1145/2688204.2688207.

[172] J. Rubin and D. Chisnell. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. Wiley-India, 2008.

[173] S. Russo Rosa. *A Framework for Experimental Validation of Domain Specific Languages*. Monte Caparica, Portugal: Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 2017.

[174] J. Sánchez-Cuadrado, J. De Lara, and E. Guerra. "Bottom-up meta-modelling: An interactive approach." English. In: *MODELS*. Ed. by R. France, J. Kazmeier, R. Breu, and C. Atkinson. Vol. 7590 LNCS. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 3–19. ISBN: 9783642336652. DOI: 10.1007/978-3-642-33666-9{_}2.

[175] V. A. Schmidt. "User Interface Design Patterns." In: 2010.

[176] P. Schmieder, B. Plimmer, and J. Vanderdonckt. "Generating systems from multiple sketched models." In: *Journal of Visual Languages & Computing* 21.2 (2010), pp. 98–108. ISSN: 1045-926X. DOI: /10.1016/j.jvlc.2009.12.003.

[177] J. Segal and C. Morris. "Developing Scientific Software." In: *IEEE Software* 25.4 (July 2008), pp. 18–20. ISSN: 0740-7459. DOI: 10.1109/MS.2008.85.

[178] B. Selic. "The pragmatics of model-driven development." In: *IEEE software* 20.5 (2003), pp. 19–25.

[179] P. Sgall, E. Hajicová, and J. Panevová. *The meaning of the sentence in its semantic and pragmatic aspects*. Springer Science & Business Media, 1986. ISBN: 90-277-1838-5.

[180] B. Shackel and S. J. Richardson. *Human factors for informatics usability*. Cambridge University Press, 1991.

[181] G. Sim and M. Horton. "Investigating children's opinions of games: Fun Toolkit vs. This or That." In: *Proceedings of the 11th International Conference on Interaction Design and Children*. ACM. 2012, pp. 70–77.

[182] H. A. Simon. *The sciences of the artificial*. MIT press, 1996.

[183] A. Sinha and C. Smidts. "An experimental evaluation of a higher-ordered-typed-functional specification-based test-generation technique." In: *Empirical Software Engineering* 11.2 (June 2006), pp. 173–202. ISSN: 1382-3256. DOI: 10.1007/s10664-006-6401-9.

[184] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. Rekdal. "A survey of controlled experiments in software engineering." In: *IEEE Transactions on Software Engineering* 31.9 (2005), pp. 733–753.

[185] D. Spinellis. "Notable design patterns for domain-specific languages." In: *Journal of Systems and Software* 56.1 (Feb. 2001), pp. 91–99. ISSN: 01641212. DOI: 10.1016/S0164-1212(00)00089-3.

[186] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[187] M. Strembeck and U. Zdun. "An approach for the systematic development of domain-specific languages." In: *Software: Practice and Experience* 39.15 (2009), pp. 1253–1292.

[188] D. Sy. "Adapting usability investigations for agile user-centered design." In: *Journal of usability Studies* 2.3 (2007), pp. 112–132.

[189] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin. "AToMPM: A Web-based Modeling Environment." In: *Demos/Posters/StudentResearch@ MoDELS*. 2013, pp. 21–25.

[190] M. Teruel, E. Navarro, V. Lopez-Jaquero, F. Montero, and P. González. "A CSCW Requirements Engineering CASE Tool: Development and usability evaluation." In: *Information and Software Technology* 56.8 (Aug. 2014), pp. 922–949. ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2014.02.009.

[191] S. Thibault. "Domain-Specific Languages: Conception, implementation and application." Doctoral dissertation. Universitè de Rennes, 1998.

[192] J. Tidwell. "Animated Transition." In: *Designing Interfaces, Patterns for effective Interaction Design* (2005).

[193] J.-P. Tolvanen and M. Rossi. "MetaEdit+: defining and using domain-specific modeling languages and code generators." In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM. 2003, pp. 92–93.

[194] A. Vallecillo, C. Morcillo, and P. Orue. "Expressing Measurement Uncertainty in Software Models." In: *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, Sept. 2016, pp. 15–24. ISBN: 978-1-5090-3581-6. DOI: 10.1109/QUATIC.2016.013.

[195] D. K. Van Duyne, J. A. Landay, and J. I. Hong. *The design of sites : patterns, principles, and processes for crafting a customer-centered Web experience*. Addison-Wesley, 2003, p. 762. ISBN: 020172149X.

[196] A. Van Lamsweerde. "Requirements engineering in the year 00: a research perspective." In: *Proceedings of the 22nd international conference on Software engineering*. ACM. 2000, pp. 5–19.

[197] A. Van Lamsweerde. "Goal-oriented requirements engineering: A guided tour." In: *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE. 2001, pp. 249–262.

[198] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. "Goal question metric (GQM) approach." In: *Encyclopedia of Software Engineering* (2002).

[199] M. Van Welie and G. C. Van Der Veer. "Pattern Languages in Interaction Design: Structure and Organization." In: *Interact'03*. Amsterdam, Netherlands: I. Press Ed., 2003, pp. 527–534.

[200] M. J. Villanueva, F. Valverde, and O. Pastor. "Involving end-users in the design of a domain-specific language for the genetic domain." In: *Information System Development*. Springer, 2014, pp. 99–110.

[201] E. Visser. "WebDSL: A Case Study in Domain-Specific Language Engineering." In: *In Generative and Transformational Techniques in Software Engineering II, Ralf Lämmel, Joost Visser, and João Saraiva (Eds.). Lecture Notes In Computer Science* 5235 (2007). DOI: 10.1007/978-3-540-88643-3{_}7.

[202] M. Völter. "Best practices for DSLs and model-driven development." In: *Journal of Object Technology* 8.6 (2009), pp. 79–102.

[203] M. Völter and J. Bettin. "Patterns for Model-Driven Software-Development." In: *EuroPLoP*. Irsee, Germany, 2004.

[204] M. Völter, C. Dietrich, B. Engelmann, M. Helander, L. Kats, E. Visser, and Wachsmuth. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform, 2013, p. 558. ISBN: 978-1481218580.

[205] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[206] R. H. Von Alan, S. T. March, J. Park, and S. Ram. "Design science in information systems research." In: *MIS quarterly* 28.1 (2004), pp. 75–105.

[207] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey. "A survey of user-centered design practice." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2002, pp. 471–478.

[208] D. M. Weiss and C. T. R. Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison Wesley Longman, Inc., 1999.

[209] R. Wieringa. "Design Science as Nested Problem Solving." In: *Proceedings of the 4th international conference on design science research in information systems and technology* (2009). DOI: 10.1145/1555619.1555630.

[210] R. Wieringa. "Empirical research methods for technology validation: Scaling up to practice." In: *Journal of systems and software* 95 (2014), pp. 19–31.

[211] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Vol. 6. Boston, EUA: Kluwer Academic Publishers, 1999.

[212] Y. Wu, F. Hernandez, F. Ortega, P. J. Clarke, and R. France. "Measuring the Effort for Creating and Using Domain-Specific Models." In: *Proceedings of the 10th Workshop on Domain-Specific Modeling*. ACM, 2010, p. 14.

[213] D. Wuest, N. Seyff, and M. Glinz. "Semi-automatic generation of metamodels from model sketches." In: *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE. 2013, pp. 664–669.

[214] E. Yu. "Towards modelling and reasoning support for early-phase requirements engineering." In: *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*. IEEE. 1997, pp. 226–235.

Appendix A

Usability Evaluation Methods

- **User Testing** - A representative amount of end users interacts with the software following a list of pre-defined tasks. Exhaustive observations of these human-system interactions allow the identification of usability issues related to the system. This evaluation method is commonly applied in a usability lab whose equipment enables the recording of user's gestures and user's computer screen for later analysis.
- **Heuristic Evaluation** - A group of usability specialists judge whether each dialogue element of the software system follows established usability principles, called "heuristics".
- **Interview** - Both end user and usability specialist participate in a discussion session about the usability of a software application.
- **User Testing – Thinking Aloud / Thinking Out Loud** - This version of user testing involves the execution of the "thinking aloud protocol". Users have to verbalize their thoughts while they interact with the software system. Supervisors should encourage end- users to express their opinions during the activity. In some cases, this indication is only requested at the beginning of the testing.
- **Usability Metrics / Software Metrics** - The purpose of this method is to establish quantitative measurements. Usability metrics quantify the usability of a system regarding effectiveness, efficiency and satisfaction. Usually, some equations are used to determine numeric values about the usability of a system. The participation of a representative number of users is required to generalize the obtained results.
- **Automated Evaluation via Software Tool** - A software tool is used to perform all the activities that are required in a usability evaluation. Depending on the type of software, this tool can be able to simulate human actions. Other applications

only keep track of the user's activities, and perform metric-based measurements. Additionally, these systems can generate a log file that can be analyzed after the testing.

- **Cognitive Walkthrough** - A usability specialist simulates the actions of a novice user of the system. During this interaction, the inspector has to identify potential issues of usability.

- **Prototype Evaluation** - Both, an end users and usability specialists participate of a meeting in which users are asked to explain their expectations about a paper prototype or a mockup.

- **Focus Group** - A representative group of end users are requested to participate in an open discussion to analyze the graphical interface of a software product. In this method, participants are free to listen and talk to other group members. In this way, they can develop own ideas based on previous comments.

- **Checklist Verification** - A usability specialist verifies if a graphical user interface meets a series of well-defined design specifications. A verification checklist helps inspectors to manage all of the details of usability that must be considered in a particular software product.

- **Pencil & Paper** - The users evaluate aspects of a prototype on paper. They are free to modify the interface design with a pencil. Additionally, they can write their comments and make annotations to specify their observations in detail.

- **Perspective Based Usability Inspection** - In each inspection session, the specialist focuses on a specific subset of usability issues covered by one of several usability perspectives. Each perspective provides the inspector a list of questions that represent the usability issues to check and a specific procedure for conducting the inspection. The assumption is that with focused attention and a well-defined procedure, each inspection session can detect a higher percentage of the problems related to the perspective used, and that the combination of different perspectives can uncover more problems than the combination of the same number of inspection sessions using a general inspection technique.

- **Field Observation / Field Study** - This method involves a usability specialist observing user's natural behavior in their "natural habitat", the field where the daily activity takes place or the workplace where the software product will be implemented. The facilitator gives the user a task and observes, takes notes, and asks questions as the user employs the software product to complete the defined task. Observation can be direct, where the inspector is present during the task, or indirect, using special software to capture user actions on the computer and record the session.

- **Eye Tracking** - This method involves measuring either where the user is looking (the point of gaze) or the motion of an eye during the use of a software product. There are several devices to perform this kind of evaluation such as: special monitors, specific cameras, sensors and even specialized software. By analyzing the visual path of the end users across the interface, it is possible to determine the relevant information, the sections that are ignored, the content which is overlooked any other gaze-related question.

- **Click Map / Scroll Map / Heat Map** - Clickmaps shows where users click on a software interface. This information allows inspectors to identify the most popular sections, and see which sections users mistake for links. This map is often represented by colors which indicate the amount of clicks in a specific area. A click map can be obtained through the use of a special software tools.

- **Opinion Mining** - This method refers to the use of natural language processing and text analysis to identify subjective information regarding the usability of a software product. For this purpose, a representative group of user have to write their opinion for certain software in three usability factors: effectiveness, efficiency and satisfaction. Then, these comments are analyzed using specialized techniques of Computer Science to determine how positive or negative are in each category.

- **Web Usability Evaluation Process** - This method involves the decomposition of the usability concept into sub-characteristics and measurable attributes, which are then associated with metrics in order to quantify them numerically. This technique has been specially developed for Web applications. The purpose is to provide feedback during all phases of the software development process. A complete model, including all the sub- characteristics attributes and their associated metrics, is provided by the authors of this method.

- **Retrospective Thinking Aloud** - This method is another variant of user testing. It is a similar practice to the thinking aloud protocol, however in this method, users have to verbalize their thoughts after the user testing session activities, instead of during them. Users are requested to use the system and perform certain tasks in silence. Participants verbalize their thoughts afterward while they are watching a recording of their performance.

- **Cognitive Task Analysis** - This techniques involves the process of learning about ordinary users by observing their interaction with a specific software product in order to understand in detail how they perform their tasks and achieve their intended goals. Tasks analysis helps to identify the tasks that a software application must support and can also help you refine the navigation or search. This method is focused on understanding tasks that require decision-making, problem-solving, memory, attention and judgment.

- **Usability Guidelines** - A group of specialists have to evaluate the graphical interface of a software product according to pre-defined usability guidelines. Although this technique is similar to heuristic evaluation, the procedure is different. In this technique, each inspector can work individually. There is no need to rate the severity and criticality of each usability issue. The assessment tool is not necessarily a set of usability heuristics. Inspectors can even use guidelines that are provided by the software development company.

- **Card Sorting** - This method can be used to verify the organization and structure of the information that appears in a software application. For this kind of evaluation, some paper cards are required. Each card has to contain a word or phrase written on one side. This expression has to represent a specific concept that is considered as part of the graphical user interface. Participants are given a stack of cards and are asked to group them together as it makes sense to them. They organize topics into categories and may also help to label these groups. If an accepted and standardized taxonomy becomes visible, it would be appropriate to apply that taxonomy in the interface.

- **Canvas Card Sorting** - This technique is a variation of the classical card sorting. This method requires that users select the most valuable concepts and arrange them in a predefined template. In this version, the main categories are previously established, and users only have to place each card into one of the groups.

- **Retrospective Sense Making** - This method is based on a retrospective protocol, in which users are asked to verbalize their thoughts after a set of tasks is completed. This specific technique establishes the use of open-ended questions in order to encourage users to process information from the long-term memory, providing justifications and explanations of certain actions they performed during their interaction with an interface. The questions should be oriented to analyze the cognitive process through which people experience problems and choose to perform certain actions, among alternative ones, in order to solve the problems experienced at a specific point in time.

- **Personas** - This method involves the description of different fictitious users of the software application. These representations should include a brief profile of goals and characteristics that represent the needs of a larger group of real users. The evaluation involves an analysis of the graphical user interface considering the goals, possible behaviors, attitudes, motivations and business objectives of each profile.

- **User Workflow** - This method establishes the elaboration of diagrams to represent all the paths that are available in a software system to perform a specific task. This diagram allows specialists to analyze the achievement of multiple goals which

involve many sub- tasks. Additionally, it permits the examination of the different users' preferences and the order in which certain tasks are performed.

- **Cognitive Jogthrough** - This method is an alternative version of the cognitive walkthrough. In this version, while inspectors are working through a series of tasks, they ask themselves a set of questions from the perspective of the user. The answers to these questions should be ranked according to the percentage of potential users are expected to have problems (from 0 to 3 in a Likert scale).

- **Domain Specific Inspection** - This method involves the use of a model that can be adapted to any software domain. Specialists should determine the areas and attributes that are more relevant for software they are going to evaluate. The inspection should be performed according to the guidelines that have been established for each usability attribute.

- **Participatory Heuristic Evaluation** - is an extension of the traditional heuristic evaluation where some principles are considered to evaluate the graphical user interface. Participatory heuristic evaluation uses the same technique. However, it involves the participation of end users in the evaluation process as "domain expert inspectors". Additionally, some additional heuristics are added to include some usability aspects that are not considered by the traditional Nielsen's proposal.

- **Semiotic Inspection Method** - The purpose of this method is the analysis of the messages conveyed through the designer-to-user metacommunication. These messages are expressed with a broad range of signs and symbols in the interface, from one or more signification systems. The aim of the semiotic inspection method is the evaluation of these elements, searching for actual or potential problems of communication and redesign opportunities to improve the communication.

- **Usability & Communicability Evaluation Method** - In this method, evaluators have to identify communication breakdowns while a representative amount of users interacts with the product software. There are thirteen expressions of communication breakdown or labels to categorize the problems of communicability and usability. The evaluator should interpret these issues and rebuild the message to identify possible improvements.

- **Simplified Pluralistic Walkthrough** - Users and designers participate together in a meeting to evaluate new ideas regarding the graphical user interface of a software product. The method does not require a working prototype. They can develop a design from just ideas. The system designers can get valuable information about the users' tasks in addition to the comments on the design.

- **Simplified Streamlined Cognitive Walkthrough** - This method establishes the same procedure than cognitive jogthough. The difference is that evaluators only

required asking two questions at each step of the inspection. Moreover, it involved to elaborate less documentation.

- **Music Performance Measurement Method** - This method establishes that the usability of a product is measured by the extent to which specific users achieve specific goals in a specific environment. Some metrics are employed to determine qualitative data regarding usability. The technique indicates that the controlled experiments should be performed as close to a real work environment. A software tool called DRUM can be used to analyze log files.

Appendix B

Patterns for DSL usability evaluation

The DSL developed through the Pheasant project (Section 9.1) is a good example of known-use of the pattern language to be illustrated in this section, as it is a complete exercise for a DSL development and is designed with strong user feedback, focusing on understanding how the language is perceived, learned, and mastered. It also gives classification of users, categorizing them by identification of their specific requirements. The case study of the language validation through Usability evaluation tests is included [18] (Annex I).

B.1 Agile development process

B.1.1 [Pattern] User And Context Model Extraction

The main goal of designing a DSL, or any other language or software system, is to satisfy the user's requirements. We need to design the language in a way that the number of user profiles covered by Usability evaluation of a DSL should be significant in relation to the actual number of intended DSL End User profiles. This means that, in the majority of cases, the number of user profiles and contexts of use characteristics will also be relevant.

B.1.1.1 PROBLEM

How to distinguish for which user profiles and contexts of use we have validated the DSL's usability level?

B.1.1.2 FORCES

- *On-Budget Completeness.* The language developers need to balance the number of features that need to be incorporated in the language and evaluation design with

the time and effort required to complete said design.

- *User Coverage*. It is sometimes easy to forget that, in general, a DSL is intended to be useful for only a relatively small set of users and not a wide range of them. When designing a language we must pay close attention not to place too much effort in satisfying requirements of non-target users.

B.1.1.3 SOLUTION

Before building a new DSL we should identify all intended user profiles and target context of use. These user groups should be characterized by their background profiles and domain expertise, as well as different stakeholder positions in solving problem groups. These general user characteristics should be weighted according to its relevance, which will influence the relevance level of each chosen test user group.

Also, in the same way we should define a complete context model that will contain all technology variations that will be possible to use, equipment availability, additional software support and its compliance to new system, as well as intended working environments and its effect on using a system.

By building a complete user and context model we are able to control for which extent of targeted user population, as well as environmental and technical range, Usability is reached. However, this is hard to achieve on a strict budget and the development team should be aware that some requirements might only be identified at later stages.

As the new domain concepts are identified for the DSL, potential users of those concepts, and contexts of use should be defined. This introduces the problem of knowing if all user groups are represented and how those user groups relate with the others and with the overall context and domain. Moreover, if usability is to be validated iteratively, the Expert Evaluator need to be able to manage and extract feedback from a large number of users on a regular basis.

For that extent, building the context and user model should be done within the domain analysis phase of the DSL development.

B.1.1.4 EXAMPLE

The user model is obtained by identifying the list of main characteristics based on which categorization of user groups is accomplished. For the case of Pheasant (see Figure B.1) these characteristics are prioritized with a Likert scale representing an evaluation importance weight ranging from 1 to 5, where 1 means 'unimportant' and 5 means 'very important'. After indicating these weights, it becomes trivial to extract important user models that need to be evaluated.

This weight hierarchy will become increasingly detailed with each new iteration. For instance, if the main profile observed is that of a physicist, we need to find details which help to isolate specific characteristics, thus creating sub profiles. In the case of Pheasant,

TECHNICAL CHARACTERISTICS				PROFILE CHARACTERISTICS				STAKEHOLDER				PERSONAL CHARACTERISTICS	
KNOWLEDGE ABOUT HEP EXPERIMENTS	5			PHYSICIST	5	experimenter	5	EXPERIMENT ROLE	5	Experiment designer	5	ANALYTICAL THINKING	5
KNOWLEDGE OF PARTICLE PHYSICS	4					theoretician	1			Analyst performer	4	LOGIC REASONING	4
KNOWLEDGE OF PROGRAMMING	3	querying	5	ENGINEER	4			ACADEMIC TITLE	4	Professor	5	SIGHT PROBLEMS	1
		c programming	4	PROGRAMMER	3					PhD	4		
		Fortran programming	2							Master student	4		
		c++ programming	2							PostDoc	3		

Figure B.1: List of user characteristics (taken from [21])

USERS WORKING EQUIPMENT						USERS WORKING ENVIRONMENT	
OFFICE COMPUTER AS WORKING PLATFORM	processor power	capacity:	2GHz-3,6GHz	number of cores	2-4	working desk	5
	RAM	capacity:	2GB-8GB			chair	5
	internal storage	capacity:	250GB-2TB	number of discs:	1-4	windows	3
	monitor	size:	20"-24"	color:	yes	office lights	4
	network	capacity:	X	Wireless ,wired	offline	air-condition system	3
	power	range:	550W-750W			heating machine	3
	office electrical power system:			secondary power:			
	keyboard:		optical				
	mouse:		optical				
OFFICE COMPUTER AS CONNECTION DEVICE TO CLUSTERED SYSTEM	processor power	capacity:	2GHz-3,6Ghz	number of cores:	2		
	RAM	capacity:	1GB-4GB				
	internal storage	capacity:	120GB-1TB	number of discs:	1-2		
	monitor	size:	20"-24"	color:	yes		
	network	capacity:	112Mbs -1Gbs	wireless, wired	online		
	power	range:	300W-550W				
	electrical power system:			secondary power:			
	keyboard:		optical				
	mouse::		optical				

Figure B.2: Users working equipment and environment (taken from [21])

we are interested in physicists who 1) have knowledge of HEP experiments and particle physics, and 2) have knowledge of programming and querying. The context model details the user's working equipment. As Pheasant is meant to be used from computers, it is essential to describe the scope of computer characteristics (see Figure B.2). This allows us to reason about whether any usability issues detected in the language can be traced to inappropriate equipment or working environment. Working environment can also cause user to obtain lower results during use of language, so it is important to describe and control main environment equipment.

Also, it is important to characterize the language operating environment to which we target the desired usability levels (see Figure B.3). As it may be too expensive to perform testing with all language operating environments configurations, one should assign different priorities for different configurations, so that at least the most important configurations are tested.

LANGUAGE OPERATING EQUIPMENT		OPERATION SYSTEM ENVIRONMENT		
Detector	1	**OS**	Linux	UNIX 5
Storage	5		Windows	Dos 3
Calculation libraries	5		Mackintosh	MAC OS 4
Robotic tape	2	**Visualization Tool**		JAS 5
Accelerator	1	**Framework**	Fortran	PAW 4
				ARTE 4
			C++	ROOT 3
				ARTE 4
				BEE 5

Figure B.3: Language operating equipment and environment (taken from [21])

B.1.1.5 RELATED PATTERNS

- *ITERATIVE USER-CENTERED DSL DESIGN* (Section B.1.3). To begin the development process, it is required that the USER AND CONTEXT MODEL EXTRACTION is featured.

- *EVALUATION PROCESS AND DESIGN PLANNING* (Section B.1.2). While the resources for the user and context model are gathered, a plan for evaluation should also be considered.

B.1.1.6 KNOWN USES

In usability testing one of the main problems for achieving usable products is that development focuses mainly on the machine or system, not considering the human aspects of software. There are three major components that should be considered in any type of human performance situation: Activity, Context and Human. Designers should focus on all three elements during development [172]. Benefits of user and context modeling on management and final product are confirmed in areas of service and interface development.

B.1.2 [Pattern] Evaluation Process And Design Planning

During the development of a software artifact such as a DSL, the development team needs to carefully plan how the development stages should proceed and what are the required features that are to be developed in each step of development. In this case, the same attention must be given to Usability evaluations and experimental designs.

B.1.2.1 PROBLEM

How to plan the processes of evaluation experiments and control the adequacy of the produced solutions to the intended users and respective context models?

B.1.2.2 FORCES

- *Planning and Control.* Through good and careful planning the engineering team becomes more able to control and validate results, and to know the scope of their impact. Planning is a time consuming task and if not done carefully induces the risk of spending resources on evaluations with questionable validity and usefulness.

- *Reusability.* Results, if packaged correctly, can be reused or replicated on another solution or similar context as long as adequate measures for each context are controlled and validated. However, it becomes easier to reason about the impact of recommendations that resulted from each experiment and reuse these conclusions for another evaluation session.

- *Balance user need validity and budget.* From the users' stand point all wished features and requirements are valid and essential. However, not all features fall within budget and not all users have the same amount of influence in the outcome and features of the DSL.

- *Experimental evaluation cost.* There is a tension between the cost of a full-blown experimental evaluation and the need to make short delivery sprints.

B.1.2.3 SOLUTION

When planning the evaluation process and experimental designs, the Expert Evaluator must document the main problem statements and their relations with intended experiments. The documentation should include initial sample modeling (considering all possible samples, groups, subgroups, disjoint characteristics, etc.), context modeling, instrumentation (e.g. type of usability tests and when to use them), the instrumentation perspectives (e.g. cognitive dimensions fundamental to assessing usability) and their relation with metrics acquired through data analysis and testing techniques.

To assess the validity of results that will lead us to reason about Usability of the domain-specific solution, Domain Experts and Language Engineers should list goals and system requirements that are basis for successful process and extent of experiments. The main problem statements and intended usability experiments should be designed with care, to ensure replicability, and to control the result of alterations.

B.1.2.4 EXAMPLE

In this pattern we need to identify and prioritize all goals of the language, as well as the goal of the evaluation. The goals for Pheasant are described in Figure B.4.

These goals will later be used to control which goals were addressed by the problem statements of experiments and the heuristic evaluations.

SYSTEM GOALS	
Deal with petabytes of data.	5
Support hundreds of simultaneous queries.	5
Return partial results of queries in progress	4
Provide interactive query refinements.	4
Deal with data on secondary and tertiary storage access for simultaneous queries	3
Support statistical selection mechanisms (uniform random sampling)	4
Provide a flexible schema which supports versioning	3
Provide an environment for data analysis that is identical on desktop workstations and centralized data repositories.	3

EVALUATION GOALS	
Query steps in Pheasant vs. the object-oriented coding	5
Aggregation	3
Expressing a decay	4
Specification of filtering conditions	3
Vertex definition and the usage of user-defined functions	5
Path expressions (navigational queries)	4
Expressing the result set	3
The expressiveness of user-defined functions	4

Figure B.4: Goal lists (taken from [21])

QUERY TASKS		USER TASKS	COGNITIVE TASKS
Run/tag selection	- Trigger selection	Inform status	Query writing
	- Run period	Write query Save query	Query reading Query interpretation
Event selection	- Filled bunch		
	- No coasting beam	Load query	Question comprehension
	- No empty events	Generate code	Memorization
	- Refined confirmation of the trigger	Undo/Redo Execute	Problem solving
Reconstruction	- Track selection	Get Query results	
	- Particle ID filter condition	Define Shema	
	- Combination of tracks	DefineUDF	
	- Vertexing	Define constants	
	- Kinematic or geometric filter conditions		
Histogramming and/or comparison with Monte Carlo Simulation			

Figure B.5: Task list (taken from [21])

Goals are fulfilled by executing tasks, therefore we need to list and prioritize them to further decide how to design instrumentation and metrics to capture these tasks (see Figure B.5)

COMPARISON ELEMENTS	
TEXTUAL VS. GRAPHIC SYNTAX	GENERAL PURPOSE VS. DOMAIN SPECIFIC
Expressive	Readability
Easy to learn	Accessibility
Syntax error Free	design reuse
Semantics error Free	high-level abstraction
Small Conceptual distance	clarity of program specification
Memorizable	program checking
Easy to use	language performance
Non-Ambigous	Maintainability
Formalizable	Portability
	Effectiveness

CAPTURE TEST
Final exams
Immediate comprehension
Reviews
Productivity
Retention
Re-learning

Figure B.6: List of comparison elements (taken from [21])

As the goal of Pheasant is to obtain better querying than in the previous approaches, it is important to list comparison elements that should be addressed during evaluation (see Figure B.6).

B.1.2.5 RELATED PATTERNS

- *ITERATIVE USER-CENTERED DSL DESIGN* (Section B.1.3). Developing an evaluation plan of action with goal and requirement analysis is an important starting point for iterative development.

B.1.2.6 KNOWN USES

Identifying and controlling evaluation process and design trough set of tasks, evaluation goals, and different test approaches is a common approach for evaluating experience in using any product or service. Examples of its use can be found in assessments of customer satisfaction, evaluation of public opinion, evaluation of psychological capabilities in human resources, as well as in evaluation of user interfaces. Detailed example of practical application to query languages can be seen in [167].

B.1.3 [Pattern] Iterative User-Centered DSL Design

When developing a new DSL, the development cycle is intertwined with scheduled deliveries of incremental versions of the DSL. Since the focus of development is usually on the delivery time and functionality, rather than the user's needs, it is usual to attain a solution which did not reached desired level of quality in use and quality of experience.

B.1.3.1 PROBLEM

How to ensure that the domain-specific solution will result in increased level of users' productivity when compared to the existing baseline?

B.1.3.2 FORCES

- *Cost of Usability Control vs. Cost of Future Modifications.* If we do not control usability tests during the several development stages, essential evaluation failures may lead us to meta-level changes that are equivalent to language development from scratch.
- *Development Cost.* Developing any language is a very expensive endeavor, more so because of the need to ensure that we will produce highly usable language that provides qualitative experience.

B.1.3.3 SOLUTION

As we discussed previously in EVALUATION PROCESS AND DESIGN PLANNING (Section B.1.2) , Productivity is related to the level of achieved Usability. Therefore, to prove the long claimed productivity increase provided by introducing DSLs, Expert Evaluator need to introduce User-Centered methods to DSL life-cycle.

On other hand, in order to increase the chances of adoption by End Users within the domain, the Language Engineers should embed User-Centered design activities within

the DSL development process itself. It is important to involve Domain Experts and End Users in the development process, empowering them to drive the project and specify their use case scenarios. However, executives and users of the language models should be involved but not overly committed to it, as users will quickly become afraid of being accountable for eventual project mishaps.

Each iteration of the development cycle should be combined with a User-Centered design activity where usability requirements are defined and validated through constant interaction with target user groups. This means that the user becomes an invaluable part of the development process and receives some measure of responsibility over the outcome of language design and development.

B.1.3.4 EXAMPLE

Like the pattern explains, we should build a schedule of all iterations at the beginning, clearly identifying participants and what features are to be tested. At each passing iteration we can then re-prioritize the remaining iterations according to what was accomplished.

These schedules should also include careful approximations of how much time and how many participants will be involved in active work on the usability evaluation. This includes the time that is required to make guidelines, list requirements, choose metrics, and implement focused workshops to discuss the results, analysis of results and so on. An example of a one such schedules is shown in Figure B.7.

ITERATION	DESCRIPTION	OUTPUTS	PERSONS		TIME
1st	heuristic analysis of implemented features with domain expert	list of typical tasks user want to perform with the language	Usability Expert	1	200h
		list of usability problems from previous cycle	Domain Expert	1-2	
		list of beneficial usability aspects from previous approach	Language Engineer	1-2	
2nd	heuristic analysis of implemented features with usability expert	checklist of usability for interfaces (ref)	Usability Expert	2-3	40h
		specification list of element structure, position, etc.	Domain Expert	1	
			Language Engineer	1-2	
3rd	usability analysis of language metamodel quality	List of language semantic clones	Usability Expert	1	60h
		List of language syntactic clones	Domain Expert	1	
			Language Engineer	2	
4th	pilot test for the first experimental evaluation with users	list of features that need to be rechecked	Usability Expert	1	120h
		list of tasks to perform with language	Domain Expert	2-3	
			Language Engineer	1-2	
5th	experimental evaluation with users following experiment design	list of detailed task and usability elements	Usability Expert	1	120h
		metrics specification	End User	14-24	
			Language Engineer	1-2	

Figure B.7: Evaluation iteration description (taken from [21])

In this case, the set of Pheasant iterations can be seen as a single development cycle step after which, if additional development was required, we would have similar usability iterations inside a new cycle with the new product in use. On this next cycle, the schedule would be easier to predict since they would be based on the numbers from the previous cycle. This gives the development team the means to control the cost of evaluation.

As expected, the 200 hours requirement of the first iteration includes the time needed to prepare and estimate the first evaluations. The following iterations require considerably less time as they are based on the previous ones.

B.1.3.5 RELATED PATTERNS

- *USER AND CONTEXT MODEL EXTRACTION* (Section B.1.1). User and context model need to be extracted so that is possible to plan which of them will have impact on iteration.
- *EVALUATION PROCESS AND DESIGN PLANNING* (Section B.1.2). Goals need to be explicitly expressed in order to plan each iteration.
- *ITERATION VALIDATION* (Section B.1.4). Each iteration should be followed by a validation stage where the output of the iteration is validated against expectations.
- *CONTEXT SCOPE TRADING* (Section B.1.5). Allows the analysis of what should be done in the next iteration.

B.1.3.6 KNOWN USES

The Usability engineering lifecycle is iterative by itself and should be merged with development of any product [143]. Involvement of user-centered techniques in iterative development of software product is becoming common, and examples vary from user interfaces to data oriented applications [53].

B.1.4 [Pattern] Iteration Validation

Developing any form of complex software artifact, the professionals in charge of development need to constantly reevaluate priorities of features and requirements according to the way the project is developing, its goals, schedules and budget.

B.1.4.1 PROBLEM

How to control which usability problems were solved, and analyze their possible relation with new ones that may arise?

B.1.4.2 FORCES

- *New features vs. fixes.* During development, it is frequent to discover new requirements that the user considers of importance. It is up to the development team to decide if these are considered new features or fixes to improve quality of solution. The latter should have top priority while the former should be carefully analyzed and sized.

- *Featurism vs. usability.* The Expert Evaluator should clearly define the line where the number of features begins to jeopardize usability rather than promoting it.
- *Loss of focus.* As the DSL development process progresses and the number of features increases, it is easy to lose track of intermediary goals. It then becomes increasingly important to validate what has been accomplished at each iteration and measure how far we are to our true goal of a usable DSL.
- *Iteration Validation schedule.* The validation itself should be short and concise, so as to not overstep the boundaries of the current iteration's development schedule. However it should be dense enough to allow the least amount of work to be postponed for additional iterations.
- *Regression Testing.* At each iteration evaluation is focused mainly on new features of the language but, as the language is growing incrementally, it ends up re-covering language details addressed in previous iterations. This is essential to ensure that new features don't deem previous features unusable, however there is also a cost associated with re-testing every previously tested feature. In this case the requirement is that at key iterations, when a new stable major version of the language is developed, testing and validation is performed on the full set of language features and not only on those newly added.

B.1.4.3 SOLUTION

Although DSLs are developed in constant interaction with Domain Experts, by validating the iterations in time-box fixed intervals we can monitor progress and check if it is going in the desirable direction. If it is not, developers are able to react to possible problems on time. At any point during language development, new requirements may arise and it is the job of the Language Engineer to evaluate them from a language point-of-view, while the Expert Evaluator is required to analyze and frame the new requirements into the time-box. The length of the project itself should not be allowed to extend over the intended deadline or to surpass the original budget except in very specific cases when the new requirements translate into make-or-break features that cannot fit into the original project scope. Nonetheless, every change in the project has to be carefully analyzed and a compromise must be reached with the decision-maker stakeholders.

If ITERATION VALIDATION is not completed at least every few iterations, when the number of features developed is enough to warrant user tests, then there is a higher risk of failure of iterative development.

Time-boxing is concluded with a progress report and with documenting results of the validations in an iteration assessment that consists of:

- A list of features that obtained the required level of usability
- A list of usability requirements that were not addressed

- A list of usability requirements that need to be reevaluated or that represent new requirement items

This should be done through explicit communication with all relevant stakeholders of the validated iteration.

B.1.4.4 EXAMPLE

Picking up Pheasant's 5th iteration from Figure B.7, validation of the iteration is accomplished by defining what features were successfully implemented and which still require some work (see Figure B.8). Understanding the status of usability evaluation for the current iteration allows us to redesign the schedule for the next few iterations.

VALIDATED	TO BE REVALIDATED	NOT ADDRESSED	ADDITIONAL FUNCTIONALITY
Expressing filter conditions	Path expressions	Environmental equipment testing	Query reuse
Expressing and using vertexing	Expressing and using UDFs	Interface design heuristics from Microsoft	Query scripting
Expressing the result set	Different data schema feature		
Expressing a decay			
Structuring the query			

Figure B.8: Iteration validation (taken from [21])

B.1.4.5 RELATED PATTERNS

- *VALIDATE ITERATIONS* ([203]). More than understanding if iterations are on track and re-working the following iterations accordingly, as the VALIDATE ITERATIONS pattern which Völter et. all. suggests, ITERATION VALIDATION requires the project team to validate if usability remains a concern throughout every iteration.
- *ITERATIVE USER-CENTERED DSL DESIGN* (Section B.1.3). Validation is a part of the iterative design and development process of a DSL.
- *CONTEXT SCOPE TRADING* (Section B.1.5). The output of ITERATION VALIDATION is fed into CONTEXT SCOPE TRADING to allow the analysis of future iterations.
- *FIXED BUDGET USABILITY EVALUATION* (Section B.1.6). Validation controls how the budget was spent to accommodate usability questions.
- *USABILITY REQUIREMENTS TESTING* (Section B.2.4). Based on requirements test results we have means to perform iteration validation.

B.1.4.6 KNOWN USES

Validating iterations of product development cycle is beneficial for controlling development of any end product. It makes clear what issues are addressed and reviles new

requirements that are overseen in planning of first cycle, and keeps track of validated approaches. This methodology helps to justify new specifications for project management and involves their decisions trough project [203].

B.1.5 [Pattern] Context Scope Trading

During the development of the DSL, the development team needs to maintain both the focus of the development and the timeline and budget set by the project owners.

B.1.5.1 PROBLEM

How to ensure that each development iteration remains focused on the user's needs while maintaining a short time frame?

B.1.5.2 FORCES

- *In-loco user.* Working directly with representative user groups, will allow detecting early the majority of usability defects so that they can be fixed at a minimum cost.
- *Following Recommendations.* Following guidelines and recommendations for the most relevant quality characteristics can be a time-consuming task. However this will result in early adoption of best practices that will eventually contribute to a usable solution.
- *User Needs vs. Project Management.* Sometimes defining requirement priorities according to user needs goes against project management best practices. It is up to the development team to ensure that both goals are achieved within the same package.
- *Sustainable focus.* When working within a budget and time limit, it is hard to focus on all usability requirements at each iteration and continue to ensure a successful iteration outcome. Some requirements are bound to receive more attention than others and lengthy requirements tend to always get pushed to future iterations [108].
- *Spread thin.* Although tempting, in medium/large projects it is impossible to take into account all intended user profiles, environmental dependencies and domain concepts in a single iteration. It is up to the engineering team to decide the iteration scope and to recognize how to profit from short iterations bursts.

B.1.5.3 SOLUTION

Short iterations require short and well scoped contexts. Each iteration needs to precisely characterize the context that specific iteration will capture from the set of global context, intended users and domain solution.

To keep the user as the focus of each agile iteration, the results of usability tests should be used to ensure that development prioritizes the most significant features, with focus on prioritized quality attributes and on the most representative user groups for the relevant context.

In order to effectively achieve this, each iteration should be preceded by a *Scope Trading Workshop* where all relevant stakeholders should come to an agreement on the context scope of the iteration. They should also agree on how the captured outcome of usability tests and experimental evaluations is to be handled.

The workshop should be used to:

- Assign a strict sequence of priorities to items in usability requirements list, depending on relevance of the domain concept's use-case;
- Identify the most relevant items from the backlog that should be solved in the next iteration;
- Reanalyze priorities of usability problems according to intended scope of user and context model;

This workshop should take place in the domain analysis phase, after validating iterations. Prior to the first iteration of the development process, identification of scope is achieved according to the extracted user and context model from the initial project plan. The intended scope of user and context model is analyzed more in depth after its definition during the workshop.

B.1.5.4 EXAMPLE

Following the scope model defined in the USER AND CONTEXT MODEL EXTRACTION (Section B.1.1) pattern, we define the User and Context scope as given in Figure B.9.

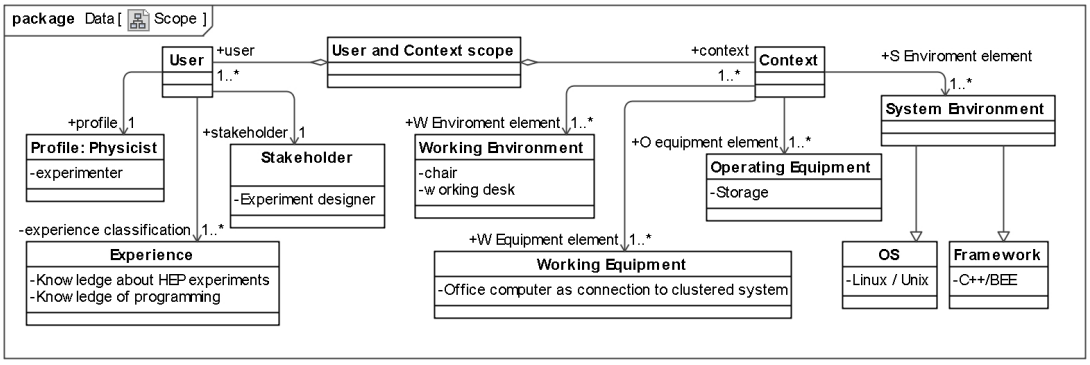


Figure B.9: Language Use Scope (taken from [21])

This scope is a subset of the scope defined in Figure B.1 and Figure B.2, accounting for the fact that changes occurred in the set of available user groups and environment

throughout the iterations. Using this new reduced scope and with the definition of evaluation for the iterations of the first cycle, as defined in Figure B.7, we define the current evaluation scope as is shown in Figure B.10.

Having defined this scope, it is easier to calculate the budget of the evaluation, and to design experimental evaluation focusing just on the given goals.

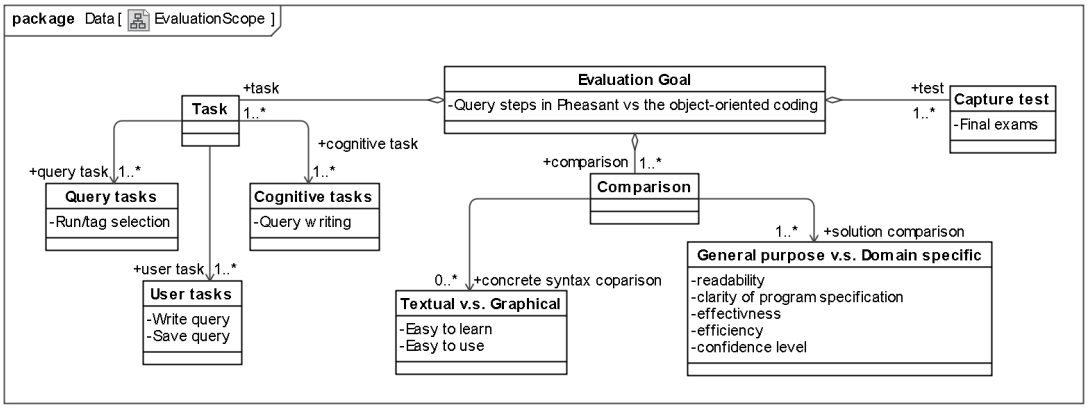


Figure B.10: Language Use Scope (taken from [21])

B.1.5.5 RELATED PATTERNS

- *SCOPE TRADING* ([203]).These patterns are very similar in idea, however it relates more to strict requirements. CONTEXT SCOPE TRADING can be seen as an extension of the original pattern to allow context trading considerations, which are valuable for DSLs.

- *ITERATIVE USER-CENTERED DSL DESIGN* (Section B.1.3). CONTEXT SCOPE TRADING is a mandatory development strategy of ITERATIVE USER-CENTERED DSL DESIGN.

- *FIXED BUDGET USABILITY EVALUATION* (Section B.1.6). The iteration scope defined within CONTEXT SCOPE TRADING constrains what can and can't be done within budget limits.

- *ITERATION VALIDATION* (Section B.1.4). The output of each validation stage is used to define what went wrong and if its solution is within budget.

B.1.5.6 KNOWN USES

Scope trading on any product development method gives input means to its budget definition [203]. Any evaluation requires precise definition of its scope, in order to be able to validate its results and indicates trade-offs in design decisions [172].

B.1.6 [Pattern] Fixed Budget Usability Evaluation

We need to develop a usable DSL for a fixed budget. The abstract nature of the language and complexity of the domain knowledge prevents contractual details from capturing every aspect that needs to be considered for a language design and implementation that leads to a system that optimally supports users in their work.

B.1.6.1 PROBLEM

How to maintain the budget within planned limits and ensure development results in a language with satisfying level of usability?

B.1.6.2 FORCES

- *Scope vs. Cost.* Evaluation, its scope and context, should be wisely planned in order to minimize its cost but provide valid usability assurance.

B.1.6.3 SOLUTION

The engineering team should regularly validate iterations to user-drive the language under construction. However, in order to reduce the cost of Usability validation in each iteration the development team should focus on:

- Using short time-board iterations that concentrate on implementing main features first and drafts of additional ones.
- Producing shippable DSLs in short iterations sprints. Since only a few features will be addressed in each iteration, the end result might have features which are left obviously unfinished and ambiguous. These unfinished features should act as motivators for user feedback.
- Getting 'live' feedback about unfinished features through brainstorming of possible solutions.
- Producing first level applications and evaluate them with users, focusing to capture usability validations related to the language design.

After each usability evaluation, Usability requirements that have failed validation must be annotated with clarifications, and listed alongside any new usability requirement that may have emerged during the last iteration. Subsequently the development team re-calculates realistic costs for all open usability requirements to enable scope trading and iteration sizing.

After a few such iterations, the work can be packaged and made available in the form of intermediary release. At this stage usability evaluation can/should be performed in real context of use with representative user groups, and language artifacts can be fully validated.

B.1.6.4 EXAMPLE

Having defined the evaluation iterations of the first evaluation cycle, presented in Figure B.7, we can calculate and fix the budget for our evaluation cycles. This budget is recalculated after each ITERATION VALIDATION (Section B.1.4). Cost estimation is made easier by having detailed cost diagrams. This enables the development team to compare the cost of each independent evaluation against the achieved result. Keeping this budget accounting also allows a more precise prediction of future costs.

INITIAL DATA	EXPECTED WORK DAYS	15 days (0-15)	5 days (16-20)	5 days (21-25)	7 days (26-32)	7 days (33-39)
	A priori Estimation	1.000 €	1.200 €	1.500 €	2.100 €	3.100 €
1ST ITERATION	Days	17	21	25	32	39
	Cost Estimation	1.050 €	1.250 €	1.550 €	2.150 €	3.150 €
	Cost Correction	+5%	+4%	+3%	+2%	+2%
2ND ITERATION	Days	17	21	25	32	39
	Cost Estimation	1.100 €	1.370 €	1.670 €	2.270 €	3.270 €
	Cost Correction	+5%	+10%	+8%	+6%	+4%
3RD ITERATION	Days	17	21	26	32	39
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.220 €	3.220 €
	Cost Correction		0%	-3%	-2%	-2%
4TH ITERATION	Days	17	21	26	32	39
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.070 €	2.970 €
	Cost Correction			0%	-7%	-8%
5TH ITERATION	Days	17	21	26	32	42
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.070 €	2.970 €
	Cost Correction				0%	0%

Figure B.11: Budget evolution for Pheasant (taken from [21])

Figure B.11 shows, for the first iteration cycle of Pheasant, how the budget evolved to encompass changes in iteration duration and cost estimation. At each passing iteration, the actual cost of the iteration was checked against the expected cost and budget corrections were made to the following iterations so that the project can be globally balanced. Having a well-balanced budget means that it becomes easier to know if the project is going according to what is expected.

One thing that must be noted in the budget of the successive iterations is that the number of expected work days also changes. This is an important fact as this indirectly influences both the monetary cost of the iteration and the scope of the following iterations.

B.1.6.5 RELATED PATTERNS

- *FIXED BUDGET SHOPPING BASKET* ([203]). It is never enough to stress that it is important to keep a fixed budget for whichever iteration style. Fixed Budget Shopping Basket details how to split the overall project development budget over all iterations.
- *CONTEXT SCOPE TRADING* (Section B.1.5). The iteration scope that is defined in turn constrains what can and cannot be done within budget limits.

- *ITERATION VALIDATION.* (Section B.1.4). The output of FIXED BUDGET USABILITY EVALUATION is used by ITERATION VALIDATION to understand if iterations are going according to plan.

B.1.6.6 KNOWN USES

This pattern represents a concrete application of a method from risk management and analysis. It is used for lowering the risks that result from big project investments and provides various advantages such as requiring the contractor to be responsible for project design and development, as well as for legacy of the projects. Applicability of these models in scheduling and cost estimation of a fixed budget that is built in construction projects is shown to be very beneficial [159].

B.2 Iterative User-Centered Design

B.2.1 [Pattern] Usability Requirements Definition

Understanding what is within the agreed budget for some project development is a skill that requires both a focus on the project and on the users' expectations by which the project's success is measured. When the main goal of the project is to achieve a usable solution, managing the users' expectations becomes much more important and can define the entire development strategy.

B.2.1.1 PROBLEM

How to define expectations and desired usability of the intended DSL?

B.2.1.2 FORCES

- *Independent perspectives on quality.* Language Engineers are able to reason about quality during development process. However, their perspective on quality does not necessarily match the perspective of other stakeholders, namely the DSLs End Users. These users originate from potentially different cultural backgrounds and have different responsibilities and motivations within the domain. That means that the perspective with which each End User of the language can look at it varies. By looking to the same language artifact, different stakeholders will mainly focus on a partial view of it, but all those partial views should be kept consistent. Features will have different importance to different stakeholders, shifting his interest to different measures of quality. Failing to identify this mismatch may lead to a solution that does not meet the expectations of the DSL users.

- *Conceptual model.* Analysis of usability requirements can bring us closer to building a correct conceptual model of solution and complete requirements model from the End Users point of view.

- *Language Choice.* When surveying commonly used software tools in the domain it is very easy to end up comparing apples with oranges. Systematic studies of the tools of the trade need to be performed, placing careful consideration with the intended use of the different tools. Tools with slightly different applicability, even if used in the same context of use should not be compared, unless the comparison takes into account these application dissimilarities. For instance, Microsoft Excel and the statistical software R can both be used to perform statistical analysis. However these are two very different tools and each excels in its own specific niche.

B.2.1.3 SOLUTION

While building domain concepts, through direct interaction with Domain Experts it is valuable to collect background information of the intended users of each language concepts, to find what usability means to them. We essentially need to have a way to keep all target user groups' needs in mind when developing the language.

The Expert Evaluator should formulate a survey, questionnaire or interview with intended user groups about their knowledge background and experience with previous approaches. This will help the engineering team to define precise user scenarios that should be the focus of the iteration cycle. While electing domain concepts, critical features that the user is concerned with should be identified and their relation with appropriate quality dimensions and attributes should be modeled. This model will later be used during experiment design to construct correct instruments, like questionnaires, to measure the distance between wished and achieved quality in use of provided solution.

In addition it is necessary to collect all data relating to the work environment and software products that are already in use to solve the problems inherent to the domain. It is important to identify characteristics that the users find that are useful, frustrating or lacking while using those products. In this way engineering team can find what quality means in the specific context of use for each user profile.

The solution provided intends to provide the basis by which the engineering team will define requirements and domain-specific goals that need to be considered. For a more in-depth explanation of this solution, we advise the reader to scan through the following example.

B.2.1.4 EXAMPLE

In the case of Pheasant, one of the main requirements that motivated the project was the need to provide a more efficient and easier to learn query language, thus overcoming the problems of the previous approach. However, the new Pheasant queries needed to remain consistent with the underlying system framework, so that would not be necessary to change previously existing queries or future queries developed in other systems. The Pheasant language needed to be developed aiming to raise the level of abstraction in such a way that the End Users could ignore individual query implementations of the different

Table B.1: Pheasant usability requirements for Understandability

Understandability	
REQ1: The language features should be easy to understand, represented with familiar notation to user	
Internal Quality	Check consistency with physics notation
External Quality	Validate ambiguous feature design decisions with Domain Expert
Quality in Use	Give simple tasks to users and capture time and eye movement in order to find required features
Quality of Experience	Capture user opinion about features that take a longer time to be assessed by user

Table B.2: Pheasant usability requirements for Expressiveness

Expressiveness	
REQ2: Provide simple way to present complex queries	
Internal Quality	Repetitive construct flows of solving complex queries should be represented near each other
External Quality	Comparison tests on effort needed to solve the same queries with different designs
Quality in Use	Measuring time needed by expert users to solve complex queries
Quality of Experience	Feedback on logical flows of provided solution
REQ3: Improved readability of queries	
Internal Quality	Check query representations of baseline approach and its problems
External Quality	Comparison tests on effort needed to solve the same queries with different designs
Quality in Use	Correctness of query interpretation by end users
Quality of Experience	Capture user suggestions of improvements for contracts that are not interpreted correctly, likability and confusions of solution representation

frameworks and in fact share their queries (i.e. have a way to talk about the specification of their queries without having to go deeply into the details of the programming environment).

Usability can be assessed at levels of Internal/External Quality, Quality in Use and Quality of Experience [103]. As follow, we present the partial list of Usability requirements and tasks for Understandability (Table B.1), Expressiveness (Table B.2), Learnability (Table B.3), Functionality (Table B.4) and Operability (Table B.5).

The diagram given by Figure B.12 shows how different internal and external quality characteristics from ISO standards influence Pheasant's Usability.

B.2.1.5 RELATED PATTERNS

- *CONCEPTUAL DISTANCE ASSESSMENT* (Section B.2.2). The requirements identified in USABILITY REQUIREMENTS DEFINITION are prioritized based on the quality attributes they impact.

Table B.3: Pheasant usability requirements for Learnability

Learnability	
REQ4: The user documentation and help should be complete	
Internal Quality	All syntactic elements of language should be well documented and consistent with metamodel change
External Quality	All given language functionalities should be explained in documentation and followed by example
Quality in Use	Check how fast is user able to perform querying using help
REQ5: The help should be context sensitive and explain how to achieve common tasks for different types of users	
Internal Quality	Check that provided description of use for each syntactic element covers all use cases that include that element
External Quality	For given use cases, check coverage of the examples provided for given language functionalities
Quality in Use	Check if the user is able to reuse same concepts in different context.
Quality of Experience	(Usually contextual help will present simple example. These should be checked with more complex examples)
REQ6: Language syntax elements should be easy to remember by the user	
Internal Quality	For each syntax element, ask the user to give it a meaning, and if it is confused to ask for other suggestion
External Quality	Provide examples on how to solve problems and ask users to solve a similar problem for which solution requires the same constructs (without consulting teaching materials).
Quality in Use	Follow how frequently users ask for help to find same concepts (operators, relation symbols)
Quality of Experience	Capture repetitive misinterpretations of language elements by novice users and provide quick test to experienced users for that element and collect feedback with additional suggestions

Table B.4: Pheasant usability requirements for Functionality

Functionality	
REQ7: Most frequent Querying task should be easy to do	
Internal Quality	Build concept element from most frequent tasks which have common logic
External Quality	Count number of steps required to perform task
Quality in Use	Measure time and number of mouse clicks/keystrokes to perform the task
Quality of Experience	Collecting feedback about likeability and pleasure that provided solution given to users
REQ8: Concepts that are parts of same task should be presented sequentially, following same logic	
Internal Quality	Sequence of domain concept relations should be analyzed against the tasks they belong to
External Quality	Make sequence diagrams with domain concepts
Quality in Use	Focus on repetitive operations of tasks and make sure they have the same use process
Quality of Experience	Collecting feedback about likeability and pleasure that provided solution given to users

Table B.5: Pheasant usability requirements for Operability

Operability	
REQ9: Language actions and elements should be consistent	
Internal Quality	Feature and behavior diagram validation with Domain Experts
External Quality	Testing if all diagram relations and rules are implemented correctly
Quality in Use	Correctness of solving tasks that are constructed based on scenarios from which diagrams were extracted
Quality of Experience	User opinion on improving consistency for tasks that have low level of correct solutions
REQ10: Error messages should explain how to recover from the error	
Internal Quality	Specifying language constructs where error recovery should be implemented
External Quality	Testing error recovery by specification
Quality in Use	Giving tasks that lead users to error messages and asking them for feedback about them
Quality of Experience	Collecting feedback about missing, misleading and incorrect error messages
REQ11: Undo should be available for most actions	
Internal Quality	specifying undo construct
External Quality	Testing of undo construct
Quality in Use	Capturing use of undo construct while solving tasks
Quality of Experience	Collecting feedback about missing, misleading and incorrect undo options
REQ12: Prevent users from producing syntax errors (e.g. misspelling)	
Internal Quality	Specifying model checkers inside the language
External Quality	Implementing and testing model checkers
Quality in Use	Capturing user's repetitive intent to produce same syntactic errors, and asking their opinion on how they can be more intuitive
Quality of Experience	Collecting syntax errors that may be produced by use of language in log files
REQ13: Prevent users from producing semantic errors	
Internal Quality	Specifying model checkers inside the language
External Quality	Implementing and testing model checkers
Quality in Use	Capturing incorrect query implementations and interviewing expert users about the given meaning (to identify cognitive problem solution or implemented meaning problems)
Quality of Experience	Capture user's frustrations of repetitive semantic errors

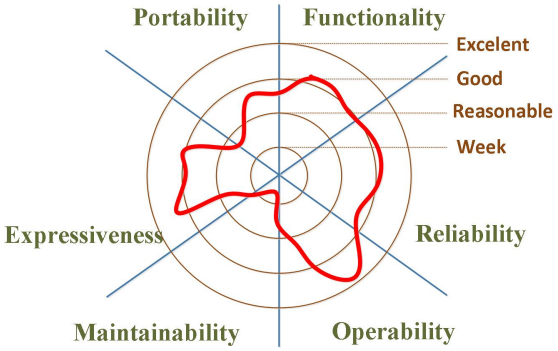


Figure B.12: Kiviat diagram of Internal/External Qualities for Pheasant (taken from [21])

- *USABILITY REQUIREMENTS TESTING* (Section B.2.4). Usability tests performed at each iteration are evaluated against the usability requirements so as to allow a definition that encompasses the current usability status of the language.

- *EXPERIMENTAL DSL EVALUATION DESIGN* (Section B.2.5). The usability requirements defined at the level of this are specified in QUALITY DESIGN MODEL that is part of EXPERIMENTAL EVALUATION MODEL.

B.2.1.6 KNOWN USES

Usability is seen as a special aspect in requirement engineering, of which the main phase is requirements definition [52]. Benefits of requirement engineering for MDD approach can be seen in examples of software product lines, supporting traceability and contributing to flexibility and simplicity in development [3].

B.2.2 [Pattern] Conceptual Distance Assessment

Extracting information from the users is a valuable source of data by which to measure the current status of our solutions. However, to be able to analyze how each requirement impacts the DSL, we need to find a way to extract influential quality attributes.

B.2.2.1 PROBLEM

How to measure conceptual distance between the user point of view to solve the problem and the provided solution?

B.2.2.2 FORCES

- *Quality Impact on Usability.* More than defining what quality attributes are important, it is essential to identify the quality attributes whose lack of actually impacts usability. That information should enable developers to produce pertinent usability metrics.

B.2.2.3 SOLUTION

In order to understand how the design of the language's architecture impacts the usability requirements, the engineering team is required to elect quality attributes and connect them with domain concepts, creating a two-way relationship of <influences/is influenced by>.

Furthermore, for each domain concept and related usability requirement, we should identify both its frequency and relevance within the domain. Weights should be assigned between the quality attributes and the domain concepts according to their influence on the final usability of the language.

Next, it is necessary to identify the frequency of different tasks that are covered by the iteration scenario. Tasks should be divided into subtasks that can be directly related with the domain concepts that will be tested.

This process will allow the Expert Evaluator to decide which usability tests are most pertinent in the current development stage and for a specific usage context. Controlling iteration priorities in turn enables a higher level of management over the usability process, by defining which usability aspects and features are to be tested iteration-wise.

B.2.2.4 EXAMPLE

For Pheasant, considering only query writing tasks, the list of subtasks that the user is required to cope with and respective frequency is as described in Figure B.13.

TASK	FREQUENCY
Inform Status	3
Write Query	5
Generate Coder	4
Execute	4
Get Query Result	4
Define Shema	3

Figure B.13: Task frequency use table (taken from [21])

Writing query task consist of four subtasks: (i) Selecting Collections, (ii) Selecting Events, (iii) Selecting the Decay and (iv) Selecting the Result. These subtasks are capturing the domain concepts presented as the metamodel elements (see Figure B.14).

After having this analysis, it makes it easier to connect the metamodel elements with usability requirements and produce concrete metrics in the terms of combination of subtasks that user need to perform.

B.2.2.5 RELATED PATTERNS

- *USABILITY REQUIREMENTS DEFINITION* (Section B.2.1). In order to consider the impact of Domain Concepts on the development, a clearly defined list of usability requirements is essential.

METAMODEL: QPHEASANT			QUERYING SUBTASK
Connectable	<--Selection		Selecting the Decay
	<--TransitionResult		
Transition			Selecting the Decay
Aggregation			Selecting the Decay
CollectionNode	<--CCOP	<--Union	Selecting Collections
		<--Intersection	
	<--CollectionSet		
	<--Exclusdion		
Event			Selecting Events
ResultNode	<--OneD		Selecting the Result
	<--TwoD		
	<--ThreeD		
	<--Histogram		
Comparison			Selecting the Decay
Distance	<--AbsDistance		Selecting the Decay
	<--RelDistance		

Figure B.14: Query subtask connection with metamodel elements (taken from [21])

- *DOMAIN CONCEPT USABILITY EVALUATION* (Section B.2.3). The impact of the domain concept on the quality of the end product influences evaluation priority and importance.

B.2.2.6 KNOWN USES

Conceptual distance has its roots in cognitive psychology. The concept of modularity that is involved in MDD allows us to measure this distance using cognitive maps [150]. Application of this approach is visible in terms of analysis of physical notations and cognitive effectiveness [41, 151].

B.2.3 [Pattern] Domain Concept Usability Evaluation

There are many advantages of determining the required quality characteristics of a DSL before it is developed and used. Metrics are a common way to determine whether a software development project is within the parameters that were defined for its execution, i.e. budget and timeline. They are also useful to analyze whether some functional goals are being accomplished. For DSL development, the focus of metric-based analysis is the language metamodel.

B.2.3.1 PROBLEM

How to capture domain concept related with usability problems using metrics?

B.2.3.2 FORCES

- *Metamodel evaluation.* The level by which a metamodel is analyzed for usability issues has a direct relation to future failures in implementation. Performing some measure of qualitative analysis of initial language metamodel, which contains the

domain concepts mapping at their initially stages, is an important step in language engineering, since problems identified at earlier phases would not be propagated onto the following phases of development.

- *Agile development.* The domain concepts defined in the language metamodel should not be considered final and can/should be analyzed at fixed stages during development in order to evaluate the ability of the metamodel to apprehend all needed domain concepts and to allow for the agile inclusion of usability requirements.

B.2.3.3 SOLUTION

During the metamodel implementation phase, which is usually complex as the Language Engineer needs to model all the domain concepts into the metamodel, it is also the time when all domain concepts are fresher and can thus be analyzed from a top-down perspective.

Using metrics to analyze metamodel concept's representation allows the engineering team to reason on how different concept modeling will impact the Usability of the DSL. Applying internal and external quality metrics we can reason about syntax dependences (i.e. metamodel's features) and their relation (i.e. meaning that they give).

Ideally the engineering team should be able to understand how changes and variations in the metamodel's design influence functionality, operability and overall usability of the language. With this knowledge he can measure and decide the importance of quality attributes to achieve the end goal and therefore which ones should be targeted and subsequently validated.

Not all metrics and measurements contribute to this end as they might not provide important feedback regarding quality improvement. The most significant metrics analyze direct DSL usage by DSL users and extract information from the gathered DSL corpus. Examples of these metrics include:

- *Clone Analysis.* Like in GPLs, duplicated code is a very well-known code smell that indicates modularization problems [34]. In DSLs corpus, more than a need to modularize, the existence of several clones, consistently showing up with a given pattern, should trigger our attention.

- *Cluster Analysis.* Identifying clusters of domain concepts in the language corpus allows the Language Engineer to evaluate if related concepts or concepts that are often used together represent a sub-language within the DSL, i.e. how the changes in the corpus are reflecting in the usability of the DSL. This is again a modularity issue, as clusters should be, as much as possible, modularly independent from other clusters, thus usability issues in one cluster should not influence other clusters.

- *Semantics-based Analysis.* Performing language analysis on the metamodel might help identify variations of the same meaning.

- *Usage Analysis.* Metamodel elements with a high level of use by the users require more thought and consideration according to usability than less used concepts.

- *Metamodel Design Pattern.* Specification of a metamodel is dependent on the designer's domain knowledge and language expertise. Thus, it is advisable to follow existing designs patterns for metamodels [57].

Careful consideration of these and other available heuristics of actual usage of the DSL will allow the development team to direct project resources to the most critical language features.

B.2.3.4 EXAMPLE

Evaluating Pheasant is not a trivial task. Nonetheless, the physicist, who takes the role of the query modeler, is immediately aware of the changes in the instances of the meta-Metamodel just by using the visual operators when modeling his query (see Figure B.15). This picture represents the direct mapping that exists from the user actions in the model to the metamodel of language.

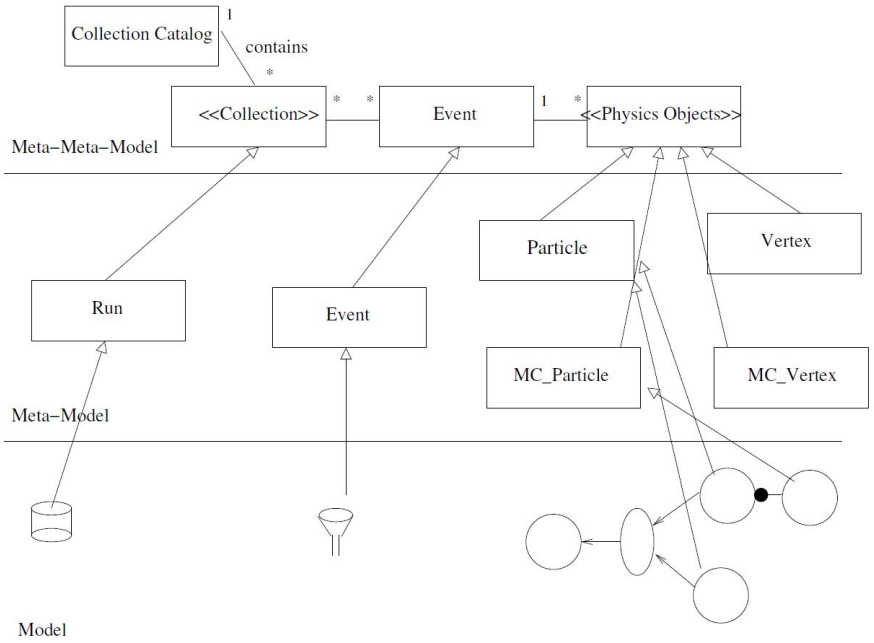


Figure B.15: Corpora relation to the metamodel tasks (taken from [7])

For the first cycles, the influence of quality characteristics of the language corpora on the user should be determined from user tasks. From these, and after the first quality assessment of the metamodel, the engineering team identifies potential need for clones and clusters. For instance, consider that the user identifies the need for two ways to accomplish the same thing, i.e. two distinct processes leading to the same outcome. The Language Engineer needs to design this in the metamodel. In this case the metamodel element representing the action needs sub-elements representing the different variations

of the same task. This need should then be validated by discussing the true impact of these clusters and clones on the language's usability. In later validations of quality in use these agreements should be tracked, so as to understand if the existing metamodel analysis premises are needed in the new version or if the scope changed.

B.2.3.5 RELATED PATTERNS

- *CONCEPTUAL DISTANCE ASSESSMENT* (Section B.2.2). The true impact of domains concepts in the quality in use of the DSL is measured by DOMAIN CONCEPT USABILITY EVALUATION.

- *USABILITY REQUIREMENTS TESTING* (Section B.2.4). DOMAIN CONCEPT USABILITY EVALUATION will also help reduce the budget for usability testing by directing tests to the most essential language features.

B.2.3.6 KNOWN USES

Evaluation of concepts is performed using a conceptual dimensions framework [124]. This approach is also used in user interfaces evaluation by building a conceptual models [107].

B.2.4 [Pattern] Usability Requirements Testing

Satisfying the user's needs should be the primary goal of a DSL. Therefore all DSLs have a strong consideration for quality in use, i.e. usability. It is important not only to define what are the principles by which the language is to be measured, i.e. which usability requirements and quality attributes define if a specific language is usable or not, but also what tests can be performed to ensure that the desired level of quality is achieved.

B.2.4.1 PROBLEM

How to analyze if the goal usability requirements are being met by the DSL?

B.2.4.2 FORCES

- *Cost of Heuristic Validation.* Heuristic validation can be a very time consuming task. However, performing non-expensive heuristic validation, we can reveal lots of relevant information about achieved level of usability.

- *Cost of User Evaluation with small number of participants.* Validation of usability with a small number of users between release cycles can identify lots of usability failures.

- *Iterative Feedback.* All feedback collected can be used to create mean values for the indicators of the next iteration cycles.

B.2.4.3 SOLUTION

At the end of each iteration, a USABILITY REQUIREMENTS TESTING stage is required to evaluate if the current implemented features go towards the usability goals previously defined [67, 143, 155].

When considering which tests to perform it is useful to consider the current state of the end product. There are usually three different levels of usability testing, depending on the current iteration:

- Initial developments or non-stable product versions should be tested by a reduced set of users, and test should be strictly focused on the features under development. Feedback can be direct, e.g. through workshops and meetings, or through small questionnaires.

- Intermediate stable versions should be tested with a group of users that are expected to interact with provided stable features. It is important to test changes and variations between stable versions and also to test if previously validated features continue to achieve the intended goal. Feedback can be collected through workshops and small questionnaires, and reused for next iterations by extensions related to additional features. At this stage it is useful to observe and analyze user's usage processes to detect small scale usability problems related to automatic tasks and cognitive processes that usually are not reported.

- Release candidates are the most important focus of usability tests. The Expert Evaluator should ensure that the users are allowed to perform the tests with a minimum of interference and constrains. If a user cannot test due to a bug in the beginning of an activity, the entire test process is undermined.

Additionally the Expert Evaluator should define, with the assistance of key stakeholders, a set of heuristic based validation methodology that will allow validation of the DSL without direct user intervention. These can be for instance a measure of user clicks to achieve a certain use case, product performance and responsiveness, ability to roll back on user errors, content placement, etc.

There are a few guidelines that should be followed to successfully perform usability tests:

- Test usability with real DSL users.

- Ideally use real usage test cases rather than dummy examples. For the final stages of development, a beta testing of a stable version of the DSL in real life usage environment should be considered.

- Tasks and features being tested should be directly related to the goals and concerns of the current iteration.

- All user feedback should be accounted for, even if no measure of importance can be given to the feedback, it might serve to provide feedback on the user's state of mind and motivations.

- If possible allow for discussion. Users usually have different views of a same subject and it is useful to allow them to debate these views in order to reach a common understanding.

B.2.4.4 EXAMPLE

Falling back to the Goal of the 5th iteration (Figure B.16), i.e. knowing how easy the language is to learn and use, usability tests are constructed following the next table.

	USABILITY MEASURES	TEST TYPES	TREATMENT
Effectiveness	- error rates while user completes querying sentences	Immediate comprehension	Learning
Efficiency	- time spent to complete a query	Reviews	Learning, Testing
Satisfaction	- confidence feedback about query	Final exams	Testing

Figure B.16: Pheasant Usability testing (taken from [21])

The testing instruments were developed as evaluation queries and feedback questionnaires.

Evaluation Queries are given in four levels of complexity. Queries are given in natural language English to be rewritten in the previously learned language (i.e. Pheasant). For each of the queries, time taken to reply them is taken. In the Pheasant project, queries were evaluated according to an error rate scale (0-5) and correctness was measured according to a self-assessment by the subject of his reply, essentially rating his feeling of the correctness of the answer. The rates were: totally correct (TC), almost correct (AC), totally incorrect (TI), not attempted (NA).

After each session, the participants were asked to judge the intuitiveness, suitability and effectiveness of the query language. After the tests are completed, the participants were asked to compare specific aspects of query languages. They rated which query language they preferred and to what extent. After the evaluation session the participants were asked to write down informal comments and suggestions for improving the language.

Example of result analysis of confidence with using the language constructs is given in Figure B.17.

B.2.4.5 RELATED PATTERNS

- *ITERATION VALIDATION* (Section B.1.4). Tests performed in USABILITY REQUIREMENTS TESTING are used to supply feedback to each ITERATION VALIDATION.

PHEASANT / BEE	NON-P	P	MEAN
Structuring the query	5/1	4/4	4.5/2.5
Different data schema feature	3.5/1	3.5/3	3.5/2
Expressing filter conditions	5/1	4.5/2	4.75/1.5
Expressing and using vertexing	5/1	5/4	5/2.5
Expressing the result set	5/1	5/3.5	5/2.25
Expressing a decay	5/1	4.5/2	4.75/1.5
Path expressions	5/3.5	3/5	4/4.25
Expressing and using UDFs	4.5/1	3.5/5	4/3
	4.8/1.3	4.2/3.9	

Figure B.17: Language constructs analysis (taken from [21])

- *USABILITY REQUIREMENTS DEFINITION* (Section B.2.1). Feedback data collected can help define next iteration usability requirements.
- *DOMAIN CONCEPT USABILITY EVALUATION* (Section B.2.3). The users' feedback provides a good starting point to define which domain concepts are correctly mapped and which pose problems.
- *EXPERIMENTAL DSL EVALUATION DESIGN.* (Section B.2.5). USABILITY REQUIREMENTS TESTING is a complementary activity to EXPERIMENTAL LANGUAGE EVALUATION DESIGN as the goals and test methodology differs.

B.2.4.6 KNOWN USES

This approach originates from usability engineering [172]. Its application can be seen in existing usability evaluation DSL examples, for instance [18, 125, 152], for longer list consult Chapter 3.

B.2.5 [Pattern] Experimental DSL Evaluation Design

Using ITERATIVE USER-CENTERED DSL DESIGN, the Expert Evaluator needs to define how to evaluate by which measure the language, or a prototype of the language, is in accordance with the elicited requirements.

B.2.5.1 PROBLEM

How to design and control the process of empirical experimentation to get sound results?

B.2.5.2 FORCES

- *Experimentation definition.* The definition of the experimentation expresses something about why a particular language evaluation was performed and may help justify the budget assigned to this type of validation [31].
- *User Expectations.* The expectations of users need to be managed and evened out prior to the experiment; otherwise there is a high chance of impact in the end result:

an extremely good result, if expectations are low or a poor result in case of high expectations.

- *User Distribution.* Ensuring that experimental evaluation is performed with an equitable distribution of users representative of the most influential groups will reduce selection bias and ensure the end results will be representative of the goal real life usage.

- *Hypothesis Guessing.* The development team through experience usually has a preconceived idea of the hypothesis result. This can influence the behavior of the experiment's participant.

- *Evaluation Scarcity.* Not all iterations require full-fledged evaluation in order for the requirements to be considered successfully achieved. However, presenting to the DSL user a final version of the language without it being thoroughly and extensively tested by DSL users in a real-life use case is not an ideal solution. Nonetheless option is used many times due to the complexities of performing experimental evaluation with DSL users.

B.2.5.3 SOLUTION

When a release candidate version of the DSL for a specific target user group seems to be ready for deployment, an experimental usability validation should be performed with real users and real test case scenarios.

Experiment planning expresses something about how it will be performed. Before starting the experiment, some considerations and decisions have to be made concerning the context of the experiment.

Only after all these details are sorted out should the experiment be performed. The outcome of planning is the EXPERIMENTAL EVALUATION MODEL (see Section 4.1.2), which should encompass enough details in order to be replicable by and independent source.

Experimental evaluation is based on quantitative evaluation of measurable properties collected from real scenarios. In this case, the aim of the experiment is to support or refute the hypothesis that the end result DSL has a direct and positive impact on usability and user performance.

B.2.5.4 EXAMPLE

The example of instantiation of EXPERIMENTAL EVALUATION MODEL is detailed in Section B.3.

B.2.5.5 RELATED PATTERNS

- *USABILITY REQUIREMENTS DEFINITION.* (Section B.2.1). The requirements defined will be validated at this stage. Also, if the development cycle is not yet complete, the feedback from EXPERIMENTAL DSL EVALUATION DESIGN is fed back into USABILITY REQUIREMENTS DEFINITION to redefine the goals of the next iteration evaluation.

- *USABILITY REQUIREMENTS TESTING.* (Section B.2.4). EXPERIMENTAL DSL EVALUATION DESIGN is a complementary activity to USABILITY REQUIREMENTS TESTING as the goals and test methodology differs.

- *EXPERIMENTAL EVALUATION MODEL.* (Section B.3). Through this pattern we are setting the processes and scope of the EXPERIMENTAL EVALUATION MODEL, OF which example pattern applications are given.

B.2.5.6 KNOWN USES

Detailed evaluation design is used in both usability engineering and experimental software engineering. This approach is modeled from the language comparison from [83] and discussed in Section 4.1.

B.3 Experimental Evaluation Model

B.3.1 [Model Instance] Problem Statement Design

Following with the example of Pheasant, we define the problem statement as a confluence of the academic context in which Pheasant is to be used. Therefore usability objectives and the experiments to measure these objectives have to take into account this context, i.e. academic level of the users, purpose, objectives and goals. This will help model a problem statement that encompasses all contextual aspects (Figure B.18).

B.3.2 [Model Instance] Context Design

The context of an experiment determines our ability to generalize from the experimental results to a wider context (Figure B.19). However, regardless of the specific context of the experiment, there are a number of context parameters that remain stable and their value is the same for all the subjects in the experiment.

B.3.3 [Model Instance] Instrument Design

Thus, having an instrument design model (Figure B.20) definition makes the task of analyzing the feedback received for target features across different iterations and users a much easier task. Modeling instruments is also useful to measure the independent

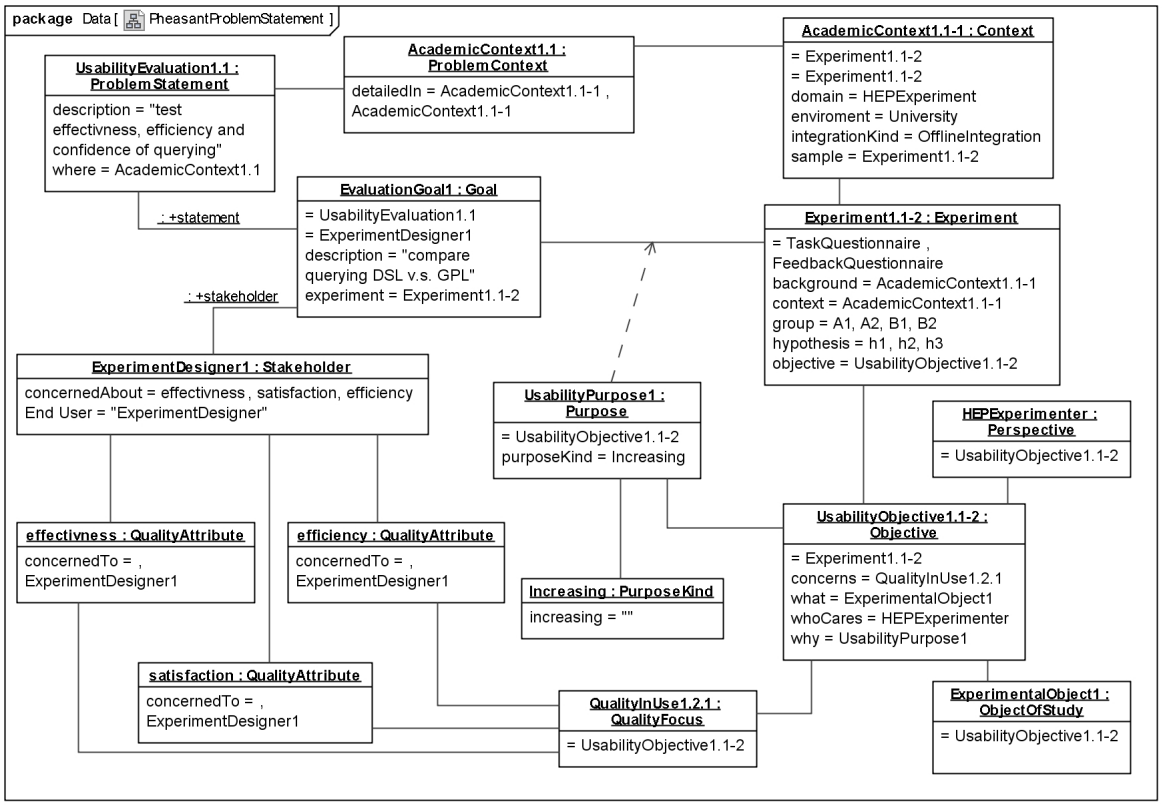


Figure B.18: Pheasant experimental Problem Statement instantiation model (taken from [21])

tasks that directly impact usability. Experimenters in human factors have developed a list of tasks to capture particular usability aspects (Sentence writing; Sentence reading, Sentence interpretation, Comprehension, Memorization and Problem solving).

For Pheasant, the Expert Evaluator defined two types of instruments for the experimentation: Task Questionnaires, designed to capture Sentence Writing, Memorization and Problem Solving, and Feedback Questionnaires, which are used to get better insight in users satisfaction, and additional recommendations.

B.3.4 [Model Instance] Sample Design

The Expert Evaluator should clearly define the profile of the participants and the artifacts that are involved in the experiment (Figure B.21).

B.3.5 [Model Instance] Quality Design

Quality focus needs to be defined through criteria, which can be recursively decomposed into sub criteria (Figure B.22). For each criterion we should specify different recommendations, i.e. positive assessments that characterize criteria. We should specify a weight for each recommendation to define which of them are more important than others for the subjects involved in the experimental evaluation.

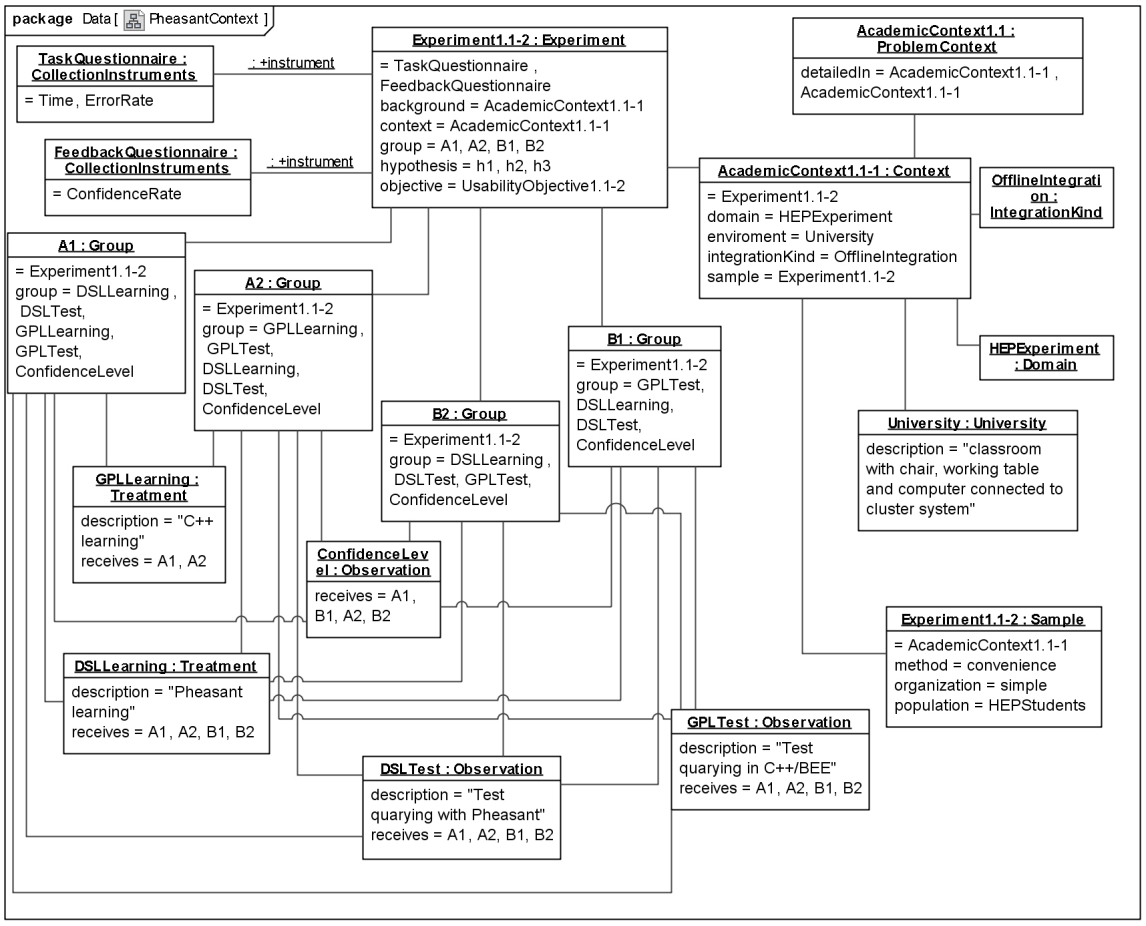


Figure B.19: Pheasant experimental Context instantiation model (taken from [21])

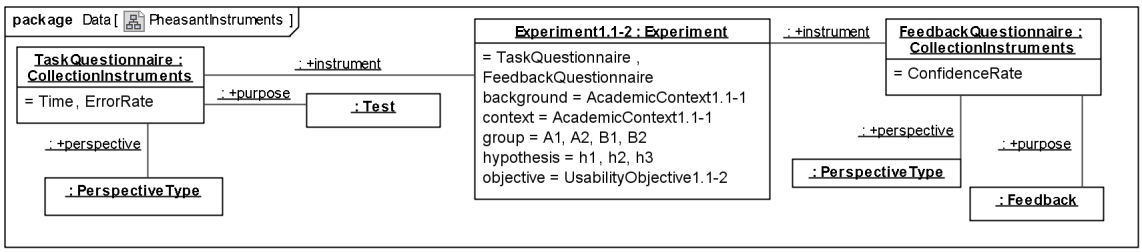


Figure B.20: Pheasant experimental Instrumental design instantiation model (taken from [21])

Evaluations of each quality criteria should be performed through methods that are specified by metrics and/or practices. Metrics gives us numerical results that can be comprised between some limits when defined, while practice can be either a pattern or an anti-pattern, applied at the process level, or on a language. Both are directly evaluated on the experiment subjects' trough recommendations [77].

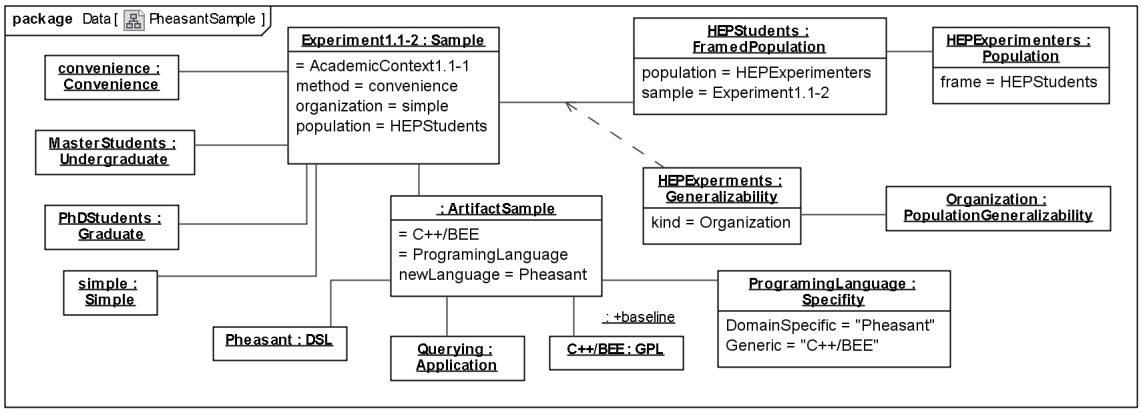


Figure B.21: Pheasant experimental Sample design instantiation model (taken from [21])

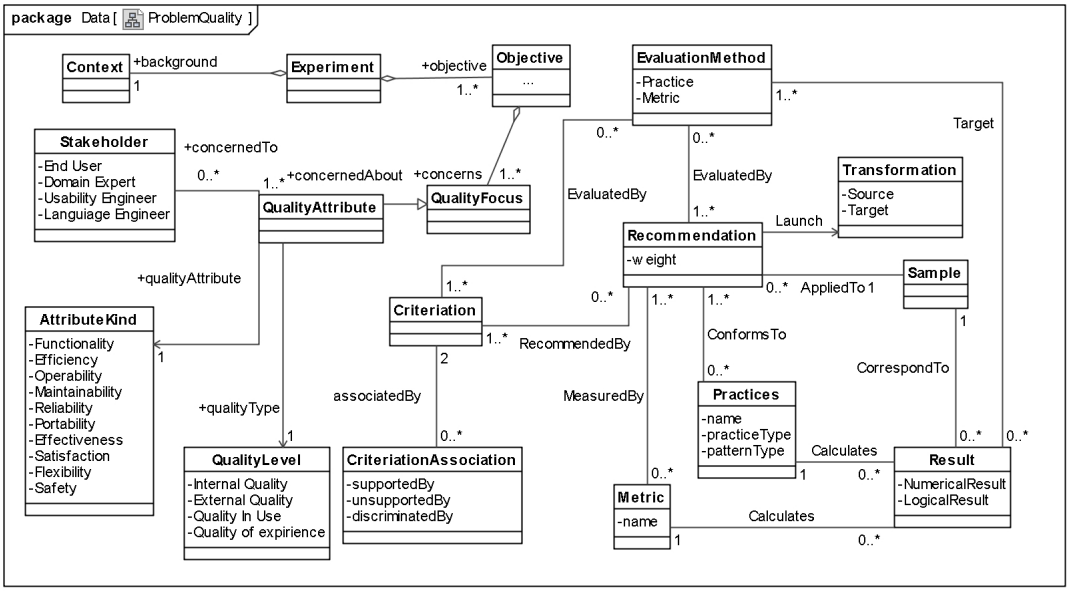


Figure B.22: Pheasant experimental Quality Design class diagram (taken from [21])

B.3.6 [Model Instance] Hypothesis and Variables Design

When a result of the evaluation does not satisfy the expected level of quality in use, the designer will need to increase the quality by setting a transformations or set of transformations. These transformations are related to language artifacts on which the evaluation was performed. Iterations can be done in same experimental settings until the desired quality is reached.

The analysis techniques chosen for the language evaluation experiment depend on the adopted language evaluation design, the variables defined earlier, and the research hypotheses being tested (Figure B.23). More than one technique may be assigned to each of the research hypotheses, if necessary, so that the analysis results can be cross-checked later. Furthermore, each of the hypotheses may be analyzed with a different technique. This may be required if the set of variables involved in that hypothesis differs from the

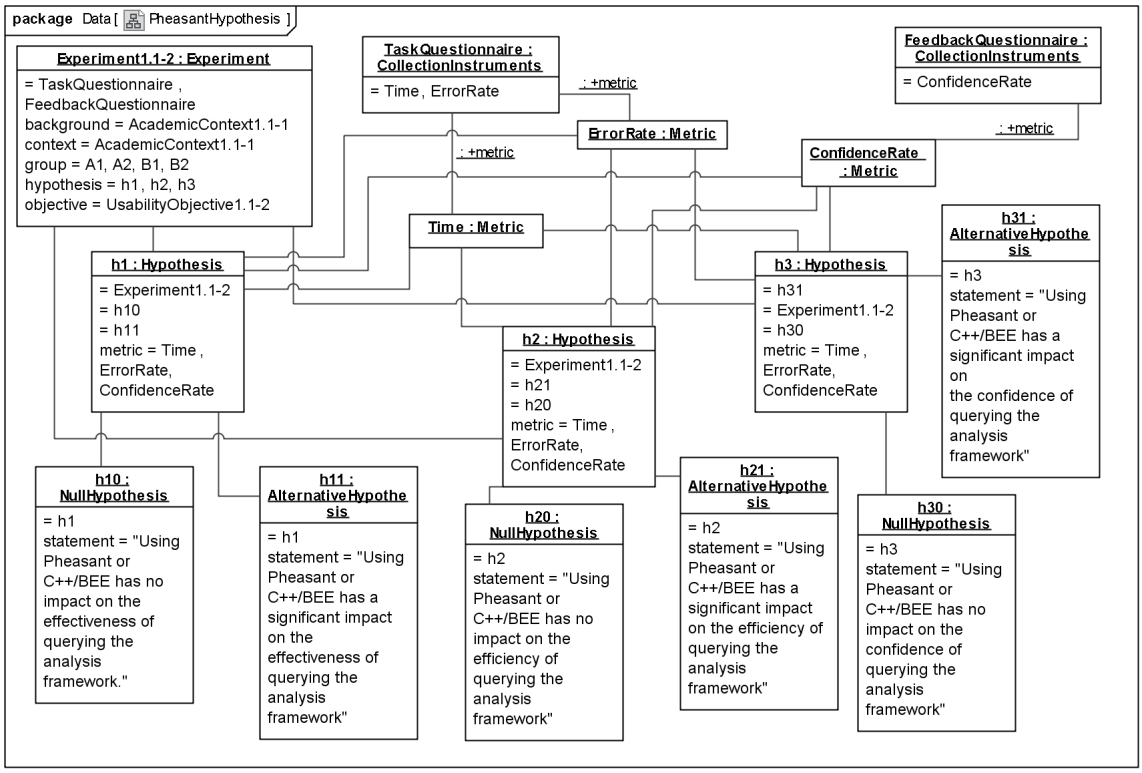


Figure B.23: Pheasant experimental Hypothesis and Variable design instantiation model (taken from [21])

set being used in the other hypotheses under tested.

ANNEX I

Case study: Pheasant

In this Annex we include the article [18] about cases study evaluation of the DSL named Pheasant (Physicist's Easy Analysis Tool), named **'Quality in use of domain-specific languages: a case study'**, which was published in Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU) at SPLASH in 2011. A detailed description of Pheasant can be found in [7].

This case study, discussed in Section 9.1, was used as a first usability evaluation example during problem investigation phase of our research process (Figure 1.1). Further, it served as illustration example for patterns (Appendix B) of pattern language which instantiate our first solution design (Section 4.2).

This work was funded by PEst-OE/EEI/UI0527/2011 Centro de Informatica e Tecnologias da Informacao (CITI/FCT/UNL).

Quality in Use of Domain Specific Languages: a Case Study

Ankica Barišić Vasco Amaral Miguel Goulão Bruno Barroca

CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Campus de Caparica, 2829-516 Caparica, Portugal
a.barisic@campus.fct.unl.pt, vasco.amaral@di.fct.unl.pt, miguel.goulao@di.fct.unl.pt, bruno.barroca@di.fct.unl.pt

Abstract

Domain Specific Languages (DSLs) are claimed to increment productivity, while reducing the required maintenance and programming expertise. In this context, DSLs usability is a key factor for its successful adoption.

In this paper, we propose a systematic approach based on User Interfaces Experimental validation techniques to assess the impact of the introduction of DSLs on the productivity of domain experts. To illustrate this evaluation approach we present a case study of a DSL for High Energy Physics (HEP).

The DSL on this case study, called Pheasant (PHysicist's EAsy Analysis Tool), is assessed in contrast with a pre-existing baseline, using General Purpose Languages (GPLs) such as C++. The comparison combines quantitative and qualitative data, collected with users from a real-world setting. Our assessment includes Physicists with programming experience with two profiles; ones with no experience with the previous framework used in the project and other experienced.

This work's contribution highlights the problem of the absence of systematic approaches for experimental validation of DSLs. It also illustrates how an experimental approach can be used in the context of a DSL evaluation during the Software Languages Engineering activity, with respect to its impact on effectiveness and efficiency.

Keywords Experimental Software Engineering, Domain Specific Languages, Usability, Language Evaluation, Software Language Engineering

1. Introduction

It is well accepted that Domain Specific Languages (DSLs) are meant to close the gap between the Domain Experts and Solution computation platforms. The general claim is that the closer we get to fill this gap, the closer we are to increase the user's productivity. The shift of the developers' focus to use abstractions that are part of the real domain world, rather than general purpose abstractions closer to the computation domain world, is said to bring important productivity gains when compared to software development using General Purpose Languages (GPLs) [11].

PLATEAU 2011 October. 2011, Portland, Oregon, USA.

Software Languages Engineering (SLE) is becoming a mature and systematic activity, built upon the collective experience of a growing community, and the increasing availability of supporting tools [13]. A typical SLE process starts with the Domain Engineering phase, in order to elicit the domain concepts. The following step is to design the language, by capturing the referred concepts and their relationships. Then, the language is implemented, typically by using workbench tools, followed by documentation. A development process goes on to the testing, deployment, evolution, recovery, and retirement of these languages. However streamlined the process is becoming, it still presents a serious gap in what should be a crucial phase: Evaluation.

If DSLs are meant to close that gap, between the Domain Experts and the Solution computation-platforms, then, from this perspective, they can be regarded as similar to **Human/Computer** (H/C) **Interaction**. The interaction should favour an increase in the efficiency of people performing their duties without this having to cause extra organizational costs, inconveniences, dangers and dissatisfaction for the user, undesirable impacts on the context of use and/or the environment, long periods of learning, assistance and maintenance [5].

Most of the requirements concerning evaluation of User Interface (UI) are actually associated with a qualitative software characteristic called *Usability*; which is defined by quality standards in terms of achieving the **Quality in Use** [8].

2. Background

The methods used to evaluate usability of GPLs are not always adequate for DSLs because they are not systematic and are centred only on computation domain concepts. The GPLs intended users are expected to have high knowledge of technical and computational concepts, while the DSLs intended user group are domain experts that are more familiar with the domain concepts. Therefore, we need a different approach to perform evaluation of DSLs.

We conducted a systematic literature review to assess the extent to which DSLs are evaluated and how they are evaluated [6]. The level of DSL evaluation found in our survey can be considered to be low, and the details on the few performed evaluations are clearly insufficient. We observed that there was a predominance of toy DSLs with unsubstantiated claims to their merits. Most authors present reports of usability evaluations that are impossible to replicate and to extract a precise rationale from, e.g., it is hard to reason about the representativeness of their DSL's users due to the poor characterization of subjects involved in the evaluation.

We were not able to find compelling evidence supporting the improvement claims on DSLs usage, with a few remarkable exceptions [14], [12]. Although this does not necessarily mean that no usability evaluation is being performed, it sends the wrong message to the practitioners who should also be concerned with the

usability evaluation of the DSLs they produce. This kind of evaluation, comparing the impact of different languages in the software development process has some tradition, in the context of GPLs, e.g. [18]), and their impact on the software developer productivity. Why should this be different with DSLs? Apparently, "some tradition" is not enough. As noted by Markstrum [15], it is also often the case where, even for GPLs, many claims on language properties (including their usability) are mostly unproven. While in this paper we are mostly concerned with DSLs and their evaluation, we regard this issue as a challenge to GPL developers, as well.

Among other possible explanations, this state of practice may stem from a lack of enough software experts that completely understand the SLE process, or from a lack of experimental evidence that clearly backs up the qualitative improvement claims that we often find in the literature. Without such evidence, it may be the case that decision makers consider proper language evaluation as a waste of time and resources. If so, they may prefer to risk using or selling inadequate DSLs rather than evaluating them properly.

The incremental nature of a typical DSL life cycle may also give the erroneous feeling that the language is being implicitly validated due to the intense interaction with the domain experts. The problem there is that the domain experts involved in the language development may not actually be the end users, and may therefore introduce biases in the perception of the language design and its usability.

Language engineers may perceive the investment in evaluation as an unnecessary cost and prefer to risk providing a solution which has not been validated, w.r.t. its usability, by end users. A good DSL is hard to build because, as noted by Mernik *et al.* [16], it requires both domain knowledge and language development expertise, and few people have both. In that case we should ask what is a cost of producing inadequate DSL for their intended users.

3. Domain Specific Languages as User Interfaces

Since we are focusing on the evaluation of usability aspects of DSLs, we need to provide a suitable definition of DSL so that we are able to evaluate them w.r.t. those usability aspects.

Intuitively, a language is a means for communication between peers. For instance, two persons can communicate with each other by exchanging sentences. These sentences are composed by signs in a particular order. According to the context of a conversation, these sentences can have different interpretations. If the context is not clear, we call these different interpretations *ambiguous*.

In our particular research we are interested essentially in the communication between humans and computers. Hence, we will only consider languages that are used as communication interfaces between humans and computers, i.e. User Interfaces (UIs). Therefore human-human languages, e.g. natural languages, and machine-machine languages, e.g. communication protocols, are not relevant for the purposes of the work described in this paper. Examples of UIs range from compilers to command-shell and graphical applications. In each of those examples we can deduce the H/C language that is being used to perform that communication: in compilers we may have a programming language; in a graphical application we may have an application specific language, and so on. Moreover, we argue that any UI is a realization of a language. A language is a theoretical object, a.k.a. model, that describes the allowed terms and how to compose them into the sentences involved in a particular human-computer communication. Languages can be deduced in two directions, human-computer and computer-human, since the feedback from the computer has to be given in such a way that it can be correctly interpreted by the humans.

Semiotics, the study of the structure and meaning of languages, is a part of linguistics that studies the dependencies and influences among *Pragmatics*, *Syntax*, and *Semantics*. The **Syntax** of a language defines what signs we can use in that language, and how we can compose those signs to form sentences. The **Semantics** of a language defines the conceptual meaning of the sentences in that language by stating how they can be logically interpreted. Finally, the **Pragmatics** of a language defines the *context of use* from which the sentences of that language can have some logical meaning.

The **Contexts of Use** i.e. *'the users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used'* [8] is one of the characteristics that we can use to evaluate DSLs usability, to pragmatically distinguish between different products: in our case different languages may have different *Contexts of Use*. Moreover, if they have different *Contexts of Use*, then we can infer that the users of those languages (the humans) most likely will have different *knowledge sets*, each one with a minimum amount of *ontological concepts* [2] required in order to actually be able to use each language.

If we say that *Context of Use* has some ontological purpose, then we can see it as *a problem to be solved* in the language user's mind. One example of this is the set of GPL where each user has to know about *programming* concepts *(variables, cycles, clauses, component, events)*, plus the domain concepts from a given *Context of Use*. Moreover, languages that reduce the use of *computation domain concepts* and focus on the *domain concepts* of the *contexts of use's* problem are called **Domain Specific Languages**. Notice that, in these pragmatic perspective languages that do not even share the same base syntax may actually share the same *domain concepts*, i.e. the intersection of their *domain concepts* is not empty for a given non-empty intersection of *contexts of use*. If the intersection of their *contexts of use* is empty then they actually do not share any of the identified *domain concepts*.

For example: consider both the SQL and C languages. The SQL language has a reserved word called $table$ to represent a database table from a DBMS. There is no $table$ in the list of reserved words of C language that the user of C can immediately read as $table$ with the same meaning as read in SQL (i.e. a database table from a DBMS). However, one of the *contexts of use* of SQL where $table$ is applied: $createtable$ can be emulated by means of a high level C (Application Programmers Interface) API function that have the same purpose of creating a table in the same DBMS. Moreover, if there is no C API supported by the DBMS, then we can even imagine how it would be to write it completely in C as part of the implementation of *the context of use* stated in $createtable$.

If we perform an analysis of the names of reusable components (in reusable infrastructures), and the reusable data structures and methods from existing APIs, and figure out all the possible ways of how they can be composed in a meaningful way then we can infer a **bottom-up** DSL from that reusable infrastructure. This **bottom-up** method of building languages by reusing existing reusable infrastructures may however generate languages that lack generality in the capability of solving any class of problems of a given domain, or if the domain of the problem is not yet fully bounded (categorized), there may be irregular composition patterns that can be nonsense w.r.t. the problem.

A **top-down** method would be to complete the domain analysis phase that is behind the existing reusable infrastructure, by discarding any existing implementation and focusing only on the complete description and categorization of the class of problems from which its users will use our new DSL to describe their solutions while using the identified *problem concepts* w.r.t. its *context of use*. If we find a mapping between all the possible expressible solutions which might be very difficult in some cases in our new DSL and the existing *concepts* of a reusable infrastructure, then we have assembled a **top-down** DSL.

DSLs that are built in a top down fashion are mostly called *horizontal* DSLs, while DSLs that are built in bottom-up fashion

are called *vertical* DSLs ([13]). In practice, it is more common for a DSL design for H/C communication to be built using a combination of bottom-up and top-down approaches.

4. Usability Evaluation

Usability is a key characteristic for evaluating the Quality of UIs, and, since we defined H/C languages as UIs, in our perspective, we should also use it for evaluating the Quality of this kind of languages. The difference between usability and the other software qualities is that to achieve it, one has to concentrate not only on system features but especially on user-system interaction characteristics. ISO 9241-11 [9] defines **Usability** as '*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*'.

ISO 9126 [8] extends this definition with the notion of *'Goal Quality'*, which has to be evaluated through the already mentioned **Quality in Use** that is perceived by the user during actual utilization of a product in its real **Context of Use**. The definition of *Quality in Use* provides a framework for a more comprehensive approach to specifying usability requirements and measuring usability with taking in account the stakeholder perspective.

Not all usability aspects can be given equal weight in a given language, so it is not always possible to achieve optimal scores for all usability attributes [3]. To evaluate the achieved Quality in Use of DSLs we find it most relevant to evaluate

- *Effectiveness* that determine the accuracy with which a developer completes language sentences
- *Efficiency* which tells us what level of effectiveness is achieved at the expense of various resources, such as mental and physical effort, time or financial cost, commonly measured in the sense of time spent to complete a sentence,
- *Satisfaction* that captures freedom from inconveniences and positive attitude towards the use of the language and
- *Accessibility* with focus on learnability and memorability of the language terms.

We need to define suitable quantitative measurements and qualitative indicators, to support a reliable assessment of the achieved quality in use. When apparently conflicting usability requirements are identified, a first approach is to look for a win-win solution that can reconcile both requirements. If this is not feasible, we need to define which usability characteristics are priority in the specific context of the project under scrutiny and favour those. These priorities can be defined based on users and tasks analysis.

To know the users we should identify the characteristics of target user population. For several kinds of end users we should analyse all kinds of them using techniques like questionnaires, interviews and observation to capture [20]: *Who* are the users?; *What* do users do?; *Why* do they do it?; *How* do they do it?; *When* do they do it?; *What* tools do they use?; Understanding 'how' and 'why' should give us deeper knowledge about the tasks. Performing task analysis by studying of the way the people perform tasks with existing systems or through high level abstraction study of cognitive processes we should identify the individual tasks the language should perform. From this we can build the desired cognitive model for language context based on user-task scenarios.

The cognitive activities involved in language are: (i)*Learning* both syntax and semantics; (ii)*Composition* of the syntax required to perform a function; (iii)*Comprehension* of the function syntax composed by someone else; (iv)*Debugging* of syntax (semantics) written by ourselves or others; (v)*Modification* of a function written by ourselves or others. Experimenters in human factors have developed a list of tasks to capture these particular aspects [19].

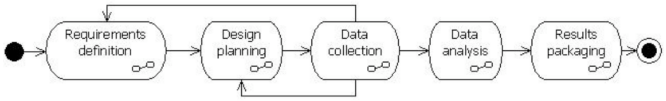


Figure 1. Experiment Activity Model Overview

Testing different tasks in the language usage is interesting, but to perform an exhaustive evaluation of them would be very expensive. Therefore, the evaluation should focus on the most critical activities.. In the case of Pheasant's evaluation, used as case study in this paper, the main concern was the task of *query writing* where users are given a question stated in natural language and have to write a sentence in the given language.

This is justified by the fact that the main function of a language is to provide is to provide users with an effective tool successfully perform some task. The goal was to know how easy it is to learn and use the language. Therefore, evaluation was restricted to three tests;

- *Immediate comprehension* - helps to identify why particular learning problems occur and they are given during teaching, immediately after some function has been taught, to determine whether the participants can use the function.
- *Reviews* - helps to identify why particular learning problems occur and they are given during teaching and cover functions taught up until that time. The participants are required to know which function to use.
- *Final exams* - tests how easily a language can be learned. These exams take place at the end of teaching the language under evaluation.

Usability evaluation is found as an important and beneficial activity in the UI development practice. It is recognized that usability must be considered from beginning of the development cycle using user centred methods. The objective of introducing user centred methods is to ensure that UI can be used by real people to achieve their tasks in the real world. This requires not only easy-to-use interfaces, but also the appropriate functionality and support for real business activities and work flows. Developing easy-to-use products makes business effective; makes business efficient; makes business sense [4]. User centred design can increase sales,reduce development, support costs and staff costs for employers.

5. Capturing achieved quality in use: a Pheasant case study

To illustrate how to evaluate the achieved Quality in Use of a DSL, we present an example of a visual query language for High Energy Physics (HEP) called Pheasant [1]. The goal of Pheasant's development was to improve the efficiency, reduce the error rate and have a less steep learning curve then the existing GPL.

The target users of the Pheasant language are specialists in HEP, with varying experience in software development. The evaluation was performed according to the mentioned ISO 9241-11 usability definition, which is an essential part of the achieved Quality in Use.

5.1 The Evaluation process

Fig.1 outlines the activities needed to perform the Pheasant language evaluation, following the scientific method. A detailed discussion on how this process can be followed in a software engineering experimentation context can be found in [7]. During *Requirements definition* problem statement (i.e. research questions), experimental objectives and context are defined. The next step is *Design planning* where context parameters and hypotheses are de-

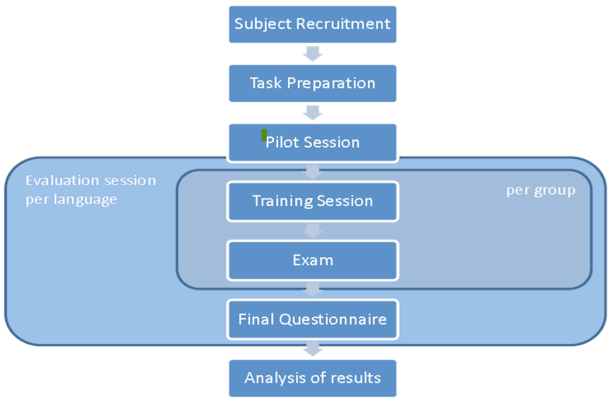


Figure 2. The evaluation process steps

fined, subjects and the sequence of observations and treatments are identified, and the data collection activities plan is set. This is followed with *Data collection*, which includes a pilot session, to correct any remaining issues, and the evaluation itself, following the designed plan. This step is followed with *Data analysis* where data is described in the form of statistical tables and graphs, and, if necessary, the data set is reduced. Hypotheses are then tested. During *Results packaging*, the results are interpreted and possible validity threats and lessons learned are identified.

The evaluation process followed in this case study is presented in *Fig.2*. The process starts with the **Participant Recruitment**, where the users are analyzed and grouped into clear categories. This way, the variables concerning the user profile that lead to different results for different groups are controlled. This step is followed by the **Task Preparation**. The aim here is to organize the evaluation by determining which tasks have to be done and which tests are elaborated in order to provide the proper results. This will generate the information required to be analyzed afterwards. The next step is the **Pilot Session**, which is meant to simulate the exam and test that the material for the training and the evaluation procedures is well organized. The main advantage of this rehearsal is to check that the time constraints and other possible external variables like proper equipment are controlled, and do not interfere with the results. Once everything is tested, we proceed to on the assessment, which we call **Evaluation Session**, for each group and language being compared. A **Training Session** is used to introduce the language. At this stage, *Immediate Comprehension* and *Review tests* are conducted with participants, while introducing the language features. The final exams, in the **Exam Session**, involve sentence writing activities. During the exam session, participants' activities are observed and recorded, so that information such as completion times and error rates can be collected. The goal is to determine the ease of learning. After each group has been evaluated in the different languages, the participants are asked for a debriefing in the form of a **Final Questionnaire Session**. The goal is to obtain the user's qualitative perspective of the comparison between the languages. In order to evaluate unbiasedly, the users should test the same environment and as realistically as possible. Evaluation process terminate with **Analysis of Results**.

5.2 Subject Recruitment

For this case study we identify two types of physicists involved, according to the context of HEP experiments:

1. *informed programmers (Inf)* are regular users of programming languages such as C, C++, Java or Fortran and they are used to program with the present analysis framework,
2. *uninformed programmers (non-Inf)* are regular users of programming languages such as C, C++, Java or Fortran and they are not used to program with the present analysis framework.

We wanted to use a third group that would consist of non-programmers but finding enough available physicist which were able to participate in this assessment turned out to be a problem. We used two different groups of programmers since the informed ones may introduce a bias on the learning phase of the compared query methodologies. This assumption is taken into account even if uninformed programmers are the majority of the population in the experiment.

At the end, fifteen graduate students were assigned to the proper group with an interview and an analysis of the participant's previous experience, to minimize the risks of biases that might otherwise be introduced by participants in a self-evaluation.

5.3 Task Preparation

Johnson [10] suggests that six individuals per subset of the population is the minimum required for a controlled experiment. Of course it is sensible to take a larger number, but the costs should be kept to a minimum. The task of gathering groups of six persons in a HEP research lab is already nontrivial. All the participants should have a degree in physics or be near its completion at least, and they should be skilled in experimental analysis. A basic knowledge of programming concepts is mandatory, since this subject is taught in the first years of the physics courses.

Introducing one query system to the whole group of participants and only afterwards the other query system would bias the evaluation, as the knowledge acquired while learning the first language would be partially reused while using the second language. In order to mitigate this threat to the validity of the results we have to split the group in two. This way, we reduce the influence of the first language while presenting the second. Mixing the two groups might lead to new variables in the evaluation that are hard to track. Therefore, we had to organize four sessions, with each group taking part in two sessions (one for each language).

The features we wanted to have evaluated are:

- query steps in Phesant v.s. the object-oriented coding
- expressing a decay
- specification of filtering conditions
- vertexing and the usage of user-defined functions
- aggregation
- path expression (navigation queries)
- expressing the result set
- the expressiveness of user-defined functions

In this study, the *independent* variables are the subject's background and the language being used. The *dependent* variables are the time to finish the task, the error rate while doing it and the confidence in the successful completion of the task.

5.4 Pilot Session

Our evaluation technique was tested with two individuals (two physics experts) in order to verify it and to test the teaching materials and questionnaires. This also helped to avoid that the evaluation had to be redone from scratch because of uncontrolled external variables, like inadequate equipment or lab conditions, or time constraints that can interfere with the results. After pilot session there was no need for significant change in experiment materials.

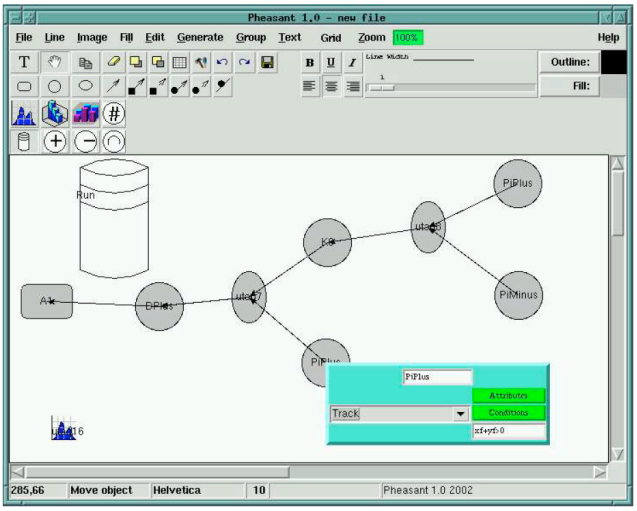


Figure 3. Query solution in Pheasant

5.5 The Evaluation Session

In the evaluation session, we try to answer the following:

RQ1: Is querying with Pheasant more effective than with C++/BEE?

RQ2: Is querying with Pheasant more efficient than with C++/BEE?

RQ3: Are participants querying with Pheasant more confident on their performance than with C++/BEE?

Our **goal** is to

- *analyze* the performance of Pheasant programmers plug-ins
- *for the purpose of* comparing it with a baseline alternative (C++/BEE)
- *with respect to* the efficiency, effectiveness and confidence of defying queries in Pheasant
- *from the point of view of* a researcher trying to assess the Pheasant DSL,
- *in the context of* a case study on selected queries.

We will test the following hypotheses:

- **H1$_{null}$** Using Pheasant or C++/BEE has no impact on the effectiveness of querying the analysis framework
- **H1$_{alt}$** Using Pheasant or C++/BEE has a significant impact on the effectiveness of querying the analysis framework
- **H2$_{null}$** Using Pheasant or C++/BEE has no impact on the efficiency of querying the analysis framework
- **H2$_{alt}$** Using Pheasant or C++/BEE has a significant impact on the efficiency of querying the analysis framework
- **H3$_{null}$** Using Pheasant or C++/BEE has no impact on the confidence of querying the analysis framework
- **H3$_{alt}$** Using Pheasant or C++/BEE has a significant impact on the confidence of querying the analysis framework

5.5.1 Training Session

Due to the complexity and the time constraints, we could not teach the complete C++ query language plus the interface of the analysis frameworks' libraries. Therefore, we focus on presenting six examples, each focusing in some of the features we chose to evaluate. The last query should make use of all the features taught in the session.

```
1)  Declare: List Runs , List Events, List Results
2)  Result is a list of particle1, particle2, Computed Vertex and Vertex
    # Step number 1
3)  while(run=nextRun()) {
4)      if(conditions) Runs.append(run)
5)  }
    # Step number 2
6)  foreach run in Runs {
7)      while(event=run.nextEvent()) {
8)          if(conditions) Events.append(event)
9)      }
    # Step number 3
10) Declare: List Particles and List Vertexes
11) foreach event in Events {
12)     Particles= event.GetParticle(conditions)
13)     Vertexes= event.GetVertex(conditions)
14)     particles=Particles
15)     While (particles.notempty()) {
16)         headParticle=particles.head()
17)         particles=particles.tail()
18)         foreach auxParticle in particles {
19)             if (condition(auxParticle) and condition(head, auxParticle)) {
20)                 Declare: distance=∞ and MinVertex={}
21)                 computedVertex=ComputeVertex(headParticle,auxParticle)
22)                 foreach vertex in Vertexes {
23)                     if(distant(vertex,ComputedVertex)<distance) {
24)                         distance=distant(vertex,Vertex)
25)                         MinVertex=Vertex
26)                     }
27)                 }
28)                 if(MinVertex.notNULL)
29)                     Result.append(headParticle,
30)                         auxParticle, computedVertex, MinVertex)
31)         }
32)     }
33) }
    # Step number 4
34) Histogram(userSetup, Results)
```

Figure 4. Query solution in C++/BEE (pseudocode based on a real query)

Murray [17] suggests that the participants should give themselves a mark for their feeling of correctness of their trial. This introduces them to the system of self-assessment. Besides, it helps the trainer to infer if there are difficulties experienced and an extra explanation is required. This session should take the time required for each group to understand the six examples.

5.5.2 Exam

We have evaluated the participants' performance in the *query writing*. Every participant has four queries, specified in English, to be rewritten in the previously learned language. An example of a query solution for the task *'Build the decay of a D0 particle to a Kaon Pion'* is given in *Fig.3* for Pheasant and *Fig.4* for C++/BEE (presented here in pseudocode, for the sake of readability)..

At the end, the subject makes a self-assessment of his reply by rating his feeling of the correctness of the answer. For each of the queries, we measured the time taken by each participant to reply in time slots of 15 minutes.

5.5.3 Questionnaire

After each session, the participants ware asked to judge the intuitiveness, suitability and effectiveness of the query language. The goal was to evaluate:

- *Overall reactions* - to obtain an overall reaction to one of the query languages through queries.

- *Query language constructs* - with the participants rating how easily specific aspects of the query language are to use.

After the tests ware completed, the participants ware asked to compare the two query languages. It is rated which query language they prefered, and into what extent.

- *Query language comparisons* - the participants are asked to compare specific aspects of both query languages and rate the preferences they have.
- *Participants' comments* - allows the participants to comment freely on the query language.

Since with the evaluation questionnaire we can only identify problems but not infer how to solve them, we ask the participants to contribute creative comments. Sometimes improvements are obvious and the comments can be fruitful. Therefore, after the evaluation session the participants are asked to write down informal comments and suggestions for improving the language.

5.6 Results analysis

In this section, we summarize the most relevant results of our evaluation tests. First, we deal with effectiveness by having a look at the test results with regard to the errors produced by the user while interacting with both evaluated approaches. Then, we will describe the results related to efficiency, which are mainly concerned with time measurements. At the end we analyse results concerning the confidence level of the participants that is measured in terms of their self-assessment.

In order to assess whether the observed differences with respect to the effectiveness and efficiency of using Pheasant, when compared to C++/BEE are statistically meaningful, we performed a Wilcoxon matched-pairs signed-rank test, as well as a sign test. These are adequate for testing our sample, as our data is ordinal. Answers ranked with 0 in correctness are used in correctness comparison (as they are meaningful to contrast the success with each of the languages - 0 means developers were not able to produce the query). However, with respect to the amount of time taken to answer, and the confidence of developers in their answer, answers ranked as 0 are considered missing answers. When they were unable to build a query, participants did not fill in the information concerning the time spent trying to build the query, nor the information concerning their confidence in their answer.

The results obtained with Pheasant were clearly better than those with the existing alternative. In order to reduce the variables that could influence the results, the queries were explained orally by an expert. This reduces the required interpretation time (which has a significant impact, especially in the group of uninformed programmers). Code re-usage was not allowed, although the subjects could use all the necessary documentation and especially the notes from the training session.

5.6.1 Effectiveness

Effectiveness and user accuracy can be assessed by observing results of the errors produced by the user while interacting with both evaluated approaches. As it can be observed in the histograms of *Fig.5*, or more detail in *Table 1*, while using C++ as a query language, the error rate was tremendous for novice users. We must state that the user did not have any sort of feedback from the system execution in order to spot the mistake and correct it before it came to the hands of the evaluator. In his daily life, the user tries to execute the algorithm and watches the result data after the execution. Then, in a cyclic way, he corrects himself and runs the query against the storage base. This is one of the main reasons why the query generation in the physics analysis phase is so time consuming. We can also observe that different groups of users get different results. As expected, their quality is directly proportional to the user's experience. Some of the most complex queries were not even tried due to the fact that they were difficult for uninformed users.

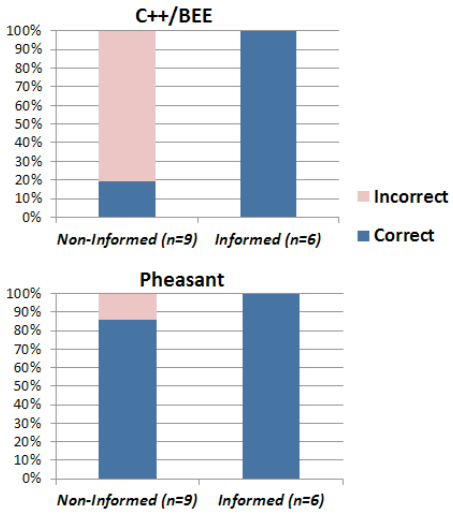


Figure 5. Effectiveness

As far as the Pheasant Query language is concerned, the results are much more promising. As the query mechanisms are much simpler and controlled, we do not observe invalid queries, and only a few wrong answers (which can be explained by some inexperience of the users in doing the analysis itself). Generally, the results show that the user did not have to essentially change the way he thinks about the query generation, which means that we have reached the goal of introducing a query language closer to the physicist's conceptual level of analysis.

Table 1. Error analysis - percent values

C++ BEE	Non-Inf	Inf
Correct	2,78	54,17
Minor data error		33,33
Minor language error	16,67	12,5
Essentially correct	*19,45*	*100*
Wrong answer	30,55	
Invalid	11,11	
Not attempted	38,89	
Totally incorrect	*80,55*	*0*
Pheasant	Non-Inf	Inf
Correct	80,5	95,83
Minor data error		
Minor language error	5,5	4,17
Essentially correct	*86*	*100*
Wrong answer	11,11	
Invalid		
Not attempted	2,89	
Totally incorrect	*14*	*0*

According to statistical analysis, presented in *Table 2*, the observed differences are statistically significant according to both tests. Also for both cases; when we analyse each query separately, as well when we look at them all together. These tests lead us to reject the null hypothesis that the obtained effectiveness is similar when using Pheasant and BEE/C++, and accept the $H1_{alt}$.

Table 2. Statistical analysis for effectiveness

	Q1	Q2	Q3	Q4	all
	Wilcoxon Signed Ranks Test				
Z	-3,097[b]	-2,714[b]	-2,949[b]	-3,037[b]	-5,833[b]
Exact Sig.	,002	,007	,003	,002	,000
	Sign test				
Exact Sig.	,000	,004	,001	,003	,000

5.6.2 Efficiency

From our time analysis in *Fig.6* and *Table 3*, it becomes clear that more time has to be spent learning and using C++/BEE than with Pheasant. This can be justified by the complexity of C++ and the BEE library. At the same time, the test participants had less confidence in the quality of his/her query. This subjective impression is confirmed, as we have seen, by the huge error rate when using BEE.

Table 3. Time analysis - percent values

		Training time *(min)*	Mean total exam time *(min)*	Mean confidence / query *(5-0)*
Non-Inf	**C++ BEE**	190	80	1,04
	Pheasant	130	65	4,75
Inf	**C++ BEE**	0	110	4,88
	Pheasant	60	60	4,83

According to the results obtained in *Table 4*, the observed differences are statistically significant according to both tests. These tests lead us to reject the null hypothesis that the obtained efficiency is similar when using Pheasant and BEE/C++, and accept the $H2_{alt}$.

Table 4. Statistical analysis for efficiency

	Q1	Q2	Q3	Q4	all
	Wilcoxon Signed Ranks Test				
Z	-2,887[a]	-3,000[a]	-2,762[a]	-2,392[a]	-5,298[a]
Exact Sig.	,004	,003	,006	,017	,000
	Sign test				
Exact Sig.	,004	,004	,004	,016	,000

5.6.3 Confidence

The test participants were supposed to rate how they were satisfied with the realization of each feature in the corresponding framework. Our goal was to identify potential weaknesses of each framework. As we can see in *Table 3*, non-Informed participants were much more confident while using Pheasant than C++/BEE. As for the informed programmers, their confidence level is almost the same with both languages. This can be regarded as a success for Pheasant. With little experience in the new language, participants felt as confident with it as with the one they were used to working with, meaning that they found the new language easy to learn.

According to the analysis presented in *Table 5*, the observed differences are statistically significant according to both test, with exception of the confidence in answering questions 3 and 4. This is likely due to a relatively higher difficulty in answering these last two questions, particularly with BEE/C++. This eventually led to the situation where uninformed programmers did not want to

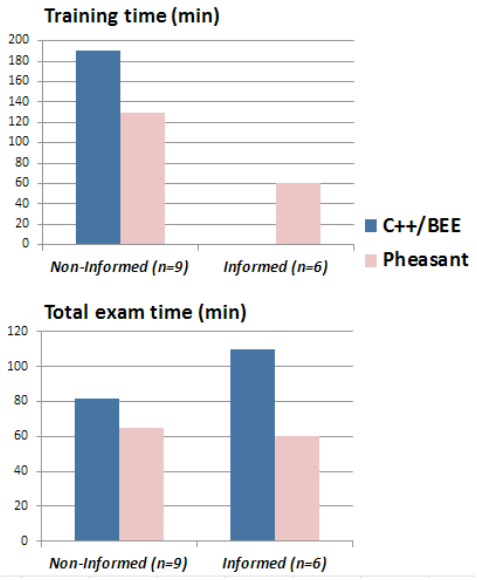


Figure 6. Efficiency

record their confidence in their answers. Because they were not able to come up with answers in BEE/C++. As we had no confidence information to compare with, for several users in these two questions and those who answered were, in general, the users with BEE/C++ expertise, the difference of confidence for these two questions follows the general trend, but is too small to be statistically meaningful. In contrast, when we aggregate the data for all questions, the advantages of using Pheasant are statistically significant. In summary, these tests lead us to reject the null hypotheses that the obtained confidence level is similar when using Pheasant and BEE/C++, and accept the $H3_{alt}$.

Table 5. Statistical analysis for confidence

	Q1	Q2	Q3	Q4	all
	Wilcoxon Signed Ranks Test				
Z	-2,232[b]	-2,232[b]	*-,966*[b]	*-,736*[b]	-3,594[b]
Exact Sig.	,026	,026	*,334*	*,461*	,000
	Sign test				
Exact Sig.	,031	,031	*1,000*	*,625*	,001

The enthusiasm towards the language was significant. The several comments focused more on implementation issues to improve interactivity and did not criticize the language itself. This is a typical situation in UIs when dealing with prototypes. It is explained by the fact that the prototype needs to evolve into the next engineering life cycle phase to result in a properly engineered software product. Only this way the product is able to provide a real analysis environment and the user can compare it in his daily life with the other alternative solutions. Although the system experts recognize that the solution is a more comfortable approach for analysis, they still worry that the query tool might be less expressive. In order to confirm or reduce these fears, we propose to carry out further tests on the feared limitations of the language, to capture if the subjects are able to write queries with the existing language constructs.

From the comments given by participants we can infer, for instance, that a query reuse mechanism should be provided in a final implementation solution. Also, a query history mechanism where

the user can browse on past queries and respective solutions, is an extra feature which might have a great impact on user satisfaction.

5.6.4 Interpretation

We have determined that, by using Pheasant, the users increase effectiveness during their query specification. It was shown that the DSL was less error-prone than the alternative, by observing that it allowed non-programmers to correctly define their queries. The evaluation also showed a considerable speedup in the query definition by all the groups of users that were using Pheasant. In general, the feed-back obtained from the users was that it is more comfortable to use Pheasant than with the alternative.

We find that the preliminary pilot study was fundamental to ensure that the subject's time was well spent. The valuable feedback of users concerning the tool support for the language, as well as their fears concerning language expressiveness support the idea of an iterative evaluation process where improvements to the language and its tool support would be performed, and then assessed in a new round of evaluation.

At this point, some legitimate questions might arise concerning to a Full-blown experimental process to evaluate a DSL, as the one we described here. Could it be that the overhead of organizing all these complex tasks is exaggeratedly too heavy compared to doing nothing? Are'nt there better, and similarly valid, lightweight alternatives to this evaluation process? Further research should be done in this direction.

6. Conclusions and Future Work

One of the main goals while producing a DSL should be to foster a more productive usage of that language by the users who will use it than the existing alternatives. The interaction should favor an increase in the efficiency of people performing their duties without this having to cause extra organizational costs, inconveniences, dangers and dissatisfaction for the user, undesirable impacts on the context of use and the environment, long periods of learning, assistance and maintenance.

Usability evaluation is most effective when it is done directly with users or in combination with expert evaluators, and the reliability of that approach usually requires lots of preparation work and a large number of people involved in it. Usability evaluation is perceived as costly and is often minimized in real-life language development processes. Nevertheless, the costs of poor usability are likely to exceed those of usability testing, in the long run. For GPLs, the user base is frequently potentially larger and more heterogeneous than the user base of a DSL so generalizing conclusions for a diverse population is harder (although, on the other hand, finding subjects for assessing a GPL is probably easier than for a DSL). Finally, it should be stressed that usability is just one of the important attributes in language evaluation. Since DSLs are built for a specific domain of use in order to close the gap between domain experts and software engineers, we find that it is essential to evaluate its usability.

Usability is one of the main quality attributes while performing UI evaluation. If we consider DSLs as a UIs, then we find that evaluating DSLs Usability can bring a positive influence on their users productivity. Moreover, unlike other software products, DSLs Usability evaluation can be an accurate activity, since precisely defined DSLs can target specific Contexts of Use, inside a particular set of user profiles.

As future work, we will propose a DSL evaluation process in the construction of new DSLs which will take into account the Usability aspect from the very beginning of their development. From this instantiation, we expect to devise languages and tools that can effectively and automatically measure the identified Usability factors early and during DSLs development.

References

[1] V. Amaral. Increasing productivity in high energy physics data mining with a domain specific visual query language. In *Phd. Thesis, University of Mannheim*, 2005. URL `http://madoc.bib.uni-mannheim.de/madoc/volltexte/2005/870/`.

[2] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20:36–41, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231149. URL `http://portal.acm.org/citation.cfm?id=942589.942704`.

[3] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. Quality in use of dsls: Current evaluation methods. In *Proceedings of the 3rd INForum - Simpsio de Informtica (INForum2011)*, 2011.

[4] N. Bevan. Cost benefits framework and case studies. *Cost-Justifying Usability: An Update for the Internet Age. Morgan Kaufmann*, 2005.

[5] T. Catarci. What happened when database researchers met usability. *Information Systems*, 25(3):177–212, 2000. ISSN 0306-4379.

[6] P. Gabriel, M. Goulão, and V. Amaral. Do Software Languages Engineers Evaluate their Languages? In *Proceedings of the XIII Congreso Iberoamericano en" Software Engineering"(CIbSE'2010)*, pages 149–162, 2010.

[7] M. Goulão and F. e Abreu. Modeling the experimental software engineering process. In *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, Lisabon, Portugal, 2007. IEEE Computer Society.

[8] International Standard Organization. Iso/iec 9126-1 quality model, June 2001. URL `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749`.

[9] International Standard Organization. Iso/iec 9241-11 ergonomic requirements for office work with visual display terminals (vdts) – part 11: Guidance on usability, June 2001. URL `http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883`.

[10] P. Johnson. *Human computer interaction*. McGraw-Hill, 1992. ISBN 0077072359 9780077072353.

[11] S. Kelly and J.-P. Tolvanen. Visual domain-specific modelling: benefits and experiences of using metacase tools. In J. Bézivin and J. Ernst, editors, *International Workshop on Model Engineering, at ECOOP'2000*, 2000.

[12] R. Kieburtz, L. McKinney, J. Bell, J. Hook, A. Kotov, J. Lewis, D. Oliva, T. Sheard, I. Smith, and L. Walton. A software engineering experiment in software component generation. In *Proceedings of the 18th international conference on Software engineering*, page 552. IEEE Computer Society, 1996. ISBN 0818672463.

[13] A. Kleppe. *Software language engineering: creating domain-specific languages using metamodels*. Addison-Wesley, 2009. ISBN 0321553454.

[14] T. Kosar, M. Mernik, and J. Carver. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering*, pages 1–29, 2011.

[15] S. Markstrum. Staking claims: a history of programming language design claims and evidence: a positional work in progress. In *Evaluation and Usability of Programming Languages and Tools*. ACM, 2010.

[16] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[17] N. Murray, N. Paton, C. Goble, and J. Bryce. Kaleidoquery–a flow-based visual language and its evaluation. *Journal of Visual Languages & Computing*, 11(2):151–189, 2000. ISSN 1045-926X.

[18] L. Prechelt. An empirical comparison of seven programming languages. *IEEE Computer*, 33(10):23–29, 2000.

[19] P. Reisner. Query languages. *Handbook of Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands*, pages 257–280, 1988.

[20] J. Rubin and D. Chisnell. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. Wiley-India, 2008.

ANNEX II

Case study: RPG DSL

In this Annex we include the article [141] about cases study in which an RPG DSL for product lines was developed, named **'The RPG DSL: A Case Study of Language Engineering using MDD for Generating RPG Games for Mobile Phones'**, which was published in Proceedings of the DSM workshop at SPLASH in 2012.

In this case study, discussed in Section 9.2, RPG DSL was completely built using MDD software development techniques and served as DSL development example during problem investigation phase of our research process (Figure 1.1).

This work was funded by PEst-OE/EEI/UI0527/2011 Centro de Informatica e Tecnologias da Informacao (CITI/FCT/UNL).

The RPG DSL: A Case Study of Language Engineering using MDD for Generating RPG Games for Mobile Phones

Eduardo Marques Valter Balegas Bruno F. Barroca Ankica Barišić Vasco Amaral

Departamento de Informatica, Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, Portugal
e.marques@campus.fct.unl.pt / balegas@live.com / mailbrunob@gmail.com / a.barisic@campus.fct.unl.pt / vasco.amaral@di.fct.unl.pt

Abstract

It is typical in the domain of digital games to have many development problems due to its increasing complexity. Those difficulties include: *i)* little code reuse in order to develop a cross-platform game; and *ii)* performing game's verification through extensive and expensive tests. This of course results in low productivity in the development (evolution and maintenance) of game solutions.

In this paper, we present a domain-specific language (DSL) for a Role-Playing Game (RPG) product lines, which was completely built using a software development technique driven by high level abstractions—called Model-Driven Development (MDD). Also, we discuss and demonstrate the several benefits of applying MDD in terms of rapid prototyping of cross-platform games, and their evaluation by means of static and dynamic verification of the game's logic properties.

Categories and Subject Descriptors H.1.0 [*Information Systems Applications*]: Models and Principles; D.2.2 [*Software Engineering*]: Design Tools and Techniques

General Terms Model-Driven Development, Domain Specific Language, Model transformation, Algebraic Petri-net

Keywords Model-Driven Development, Domain-Specific Language, Model transformation, Algebraic Petri-net, Role Playing Games, Game Analysis

1. Introduction

The increasing complexity of software development– mostly due to the increasing complexity of the Functional and Non-Functional requirements involved (problem domain), and the supporting platforms and technology (solution domain)– has been the main challenge of software engineering as research topic since its origins. The lack of reuse for the new solutions [3, 6] and the lack of infrastructures that allow rapid development to improve the development life-cycle in what concerns to requirements' validation, have been some of the main reasons and consequences of the poor quality of Software Projects. Software game development is no exception: the process of game development is usually associated with low productivity—it may take years to see the final product. This is due the fact that the developed games have complex graphics, logic, artificial intelligence and input devices [2]. Their validation is performed through extensive and costly tests, and most of them are cross-platform. This last characteristic means however that there is a potential gain in reusing the produced software game not only to reduce development costs, but also to reduce verification's and validation's costs.

In the Game Domain, we observed that games can be organized into a wide range of categories each one sharing common game logic. For instance, in the category of Role-Playing Games (*RPGs*), all games tend to share the same concepts (e.g., characters, dialogues, maps and quests), regardless of the underlying implementation technology. Again, this characteristic is a potential target for code reuse at the game logic level. Therefore it makes sense to build a Domain Specific Modeling Language (*DSML*) to design, and deploy RPG games.

The work of software language engineers [10] is to develop languages that are able to provide those abstractions to the Experts in a given Application Domain — or in our case a Game Designer. These languages must be simple, focus on the domain of the problem, and use a vocabulary that is natural to the domain expert. As such, software language engineers can use MDD techniques to develop languages by having models of both the language's syntax (by means of Meta-models), and the language's semantics (by means of transformation models).

In this paper, we present an example of SLE (Software Language Engineering) while applying MDD techniques, where we engineer a visual DSML for a Role-Playing Game (*RPG*) product line (we can create different RPG games using the same DSML). The end products (each individual game) is deployed in a smartphone platform. With this language, the Game Designer is able to model a game in terms of rules, challenges, characters, etc., and generate the code for a given target platform. It is possible to build a series of RPG games with diverse features, creating this way different types of games: one may have only mazes to solve; another may have agents to interact; etc. In addition, we can perform analysis by using Model Checkers[1], verifying properties on the designed games such as: it is possible to finish the specified game; or that it is possible for a player to get the maximum score (in w.r.t. the score definition in the specified RPG game).

2. The Approach

In this section we discuss the tools used to implement our solution [1] and show a complete workflow to generate RPG games and verify them. This methodology is reusable and can be applied to any

DSM Workshop'12 22nd October 2012, Tucson, Arizona, USA.

[1] For more details, our solution is available online at http://solar.di.fct.unl.pt/twiki5/pub/Projects/BATIC3S/ReleaseFiles/RPGCaseStudy.zip

type of MDD project. The process is divided in three stages: Domain Analysis, Design, and Implementation. Briefly, in the Domain Analysis phase we define the domain of the application being developed and the domain for the target platform, on which we want to deploy the solution. In this phase, it is also important to develop a vocabulary that is easy for the domain experts to use. In the Design phase, we precisely describe the previously analysed domains. These descriptions (or models) are the input for the Implementation phase. Finally, in the Implementation phase, we realize (by means of transformation models) the models resulting from the Design phase into concrete artifacts in a target platform—let it be an execution or an analysis platform, or in our particular case: both.

2.1 Domain Analysis

In the Domain Analysis phase we worked in two different levels, at the level of the problem (i.e., expressing the concepts and logics of RPG Games), and at the level of the solution (i.e., how those concepts can be realized in a computational platform). In the level of the problem of RPGs' design, we tried to express and define what would be the common characteristics of all RPG games, regardless of what are the requirements of their implementation on an underlying computation infrastructure. In the following subsections, we describe the domain that we analysed for the RPGs. From this analysis, we define our DSML's syntax by means of a Meta-model.

2.1.1 Concept Agents

Agents are the characters of an RPG that occupy some cell in a scene, and with whom the hero may interact. The agents have attributes, inventory, resources and a set of actions. There is a broad variety of attributes such as strength, agility, intelligence, health points (these are mandatory on every game) or magic points. The health points may be recovered using items, or recovered with time, but if they reach 0, the agent dies. Also, an agent has an inventory where all the items, resources and equipment, gathered by him/her, are kept. Finally, the agents' actions can be specified using our RPG language. For instance, in dialogues, the phrases said by the agent are determined with a decision tree, but, during a combat, the selected movement is randomly selected in a set of possible fighting movements. The possible actions of agent are to walk, to fight, to talk and to give, buy or sell items. When an agent dies it automatically disappears from the map, and it may leave behind some items in its place, or give some resources to the hero.

2.1.2 Concept Hero

The hero is the controllable character of any RPG — i.e., it is the agent that the human player controls with a broader set of actions. When the hero dies, the game ends. The possible set of actions available to a hero, in our RPG game language, are: to move between cells, to interact with other agents by talking or fighting them, to interact with items and objects by picking them up, giving, buying or even selling them. Resources, such as gold, may be used to buy other items, these are gained throughout the scenes or by fighting hostile agents. All of the specified RPG benefit from an experience system, where a hero's abilities improve through the accomplishment of objectives and interaction with other agents or object. A hero can gather a small amount of experience points by accomplishing those tasks and experience points can be spent to improve the hero's attributes. Also, it is possible to equip some items of the inventory, that can improve some of the attributes of the hero, as long as they are equipped. The inventory can be checked or modified at any time. The hero attributes can also be modified by state conditions (e.g., poisoned, burned or sleepy).

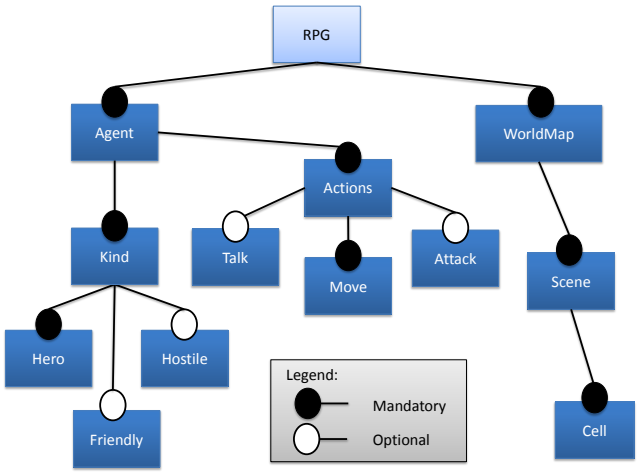


Figure 1. Excerpt of the RPG Games feature model.

2.1.3 Concept Space

Every game has a world map, which is the environment where the game takes place. The world map is composed by many different scenes, which are connected, and the agents can move across. A scene contains a two-dimensional map of cells. The agents can move between cells if they are unoccupied. In the map there exist objects or artifacts to be picked up, traps that cause the agents to lose health points, switches that, when activated, let the hero progress to another scene, and doors that allow the passage of agents to other scenes.

2.1.4 Concept Objectives

Each game has a main objective, that once finished ends the game, and there may also be other objectives, which are considered secondary objectives. There are different kinds of objectives: interaction with agents (e.g., talk with a specific agent), get to a specific scene (e.g., arrive at castle), or get artifact (e.g., get golden cup). The number of objectives completed determines the final score of the game. In Figure 1, we show a piece of the feature model for the RPG language. The feature model expresses the features that are mandatory or optional and also the relations between them. We used the feature model notation to represent the variability of RPG games schematically. Here we can see how we modelled the space elements defined in the Domain Analysis phase.

With this complete feature model for RPG games we derived our RPG DSL's syntax (by means of an Ecore Meta-model) based on the relations discussed above.

2.1.5 Execution Platform

In the Domain Analysis of the solution (computational) level, we had to choose platforms to both deploy the generated games and to analyse them. Since we did not have any kind of experience in this area, nor any base prototype to work on, we had to choose a game developing platform and build our own framework on top of it. So we analysed some of the existing game engines that could be useful to implement our RPG Games. The criteria to choose the target framework were: fast development; abstraction level relatively to system calls and hardware dependencies (e.g., graphical primitives, input modalities, etc.); need of previous knowledge in the area of game engines.

We analysed three frameworks and identified some of its characteristics that we considered advantageous in the use of each framework. In the table 1 we describe these.

Table 1. Comparative table between frameworks

Framework	characteristics
Slick	Java based; Uses LWJGL
Sphere	Scripting language; Level of abstraction that allows some of the typical features of RPGs
Corona	Scripting language; Allows cross-platform compilation for Android and iOS

Between these three frameworks we chose Corona SDK [2] because it seemed interesting to allow compiling the game for different mobile platforms. The game development is done in the Lua language: a scripting language, which is preferred for rapid development [18].

2.1.6 Analysis Platform

With regard to the choice of the analysis platform, we had first to analyse what are the analysis requirements for our RPG game specifications. As in every game development it is possible to create games that are impossible to finish, so it is expected that the games automatically generated in our approach have models in the DSL that comply to some desired properties. We used OCL [8] to perform static analysis over our RPG models. This kind of verification assures that models are well formed and therefore can be used through the MDD cycle. However, they cannot guarantee that dynamic properties are valid over all possible computation steps (also known as symbolic states) of the game. In particular, in our RPG Game Language, we want to assure that all developed games have the possibility of *i)* finishing a game, and *ii)* finish a game with the maximum score.

To check this kind of properties we integrated a model checker[1] in our DSML. There are various model checkers in the literature[9, 12, 13], but we choose ALPiNA [4], an Algebraic Petri-net (APN) [11] analyser. Although this tool suffers (as other model checkers) from the exponential nature of the model checking problem, where the analysis space tends to explode with the size of the problem [5], this tool actually presents good performance while checking invariants in Petri net models. Also its APN models are expressed with the same ECore format as the models we use to define new RPG games, therefore, we could integrate the code verification in the MDD cycle seamlessly using this tool, by performing the transformation from our RPG model's to APN model's and by analyzing the latter ones with ALPiNA.

2.2 Design

After making an overview of the requirements of our language we defined the required Meta-models to implement it. We used Ecore-based Meta-models used by the Eclipse Modeling Framework (EMF) [16] [3]. The Meta-models describe the RPG Language and the abstractions of the target platforms. This is the base of the whole process of deploying the RPGs. A bad design of Meta-models may lead to severe changes in its implementation, execution, analysis and graphical editors. In our project we created three Meta-models, one for the main language from which we are able to

[2] http://www.anscamobile.com/corona/

[3] http://eclipse.org/modeling/emf/

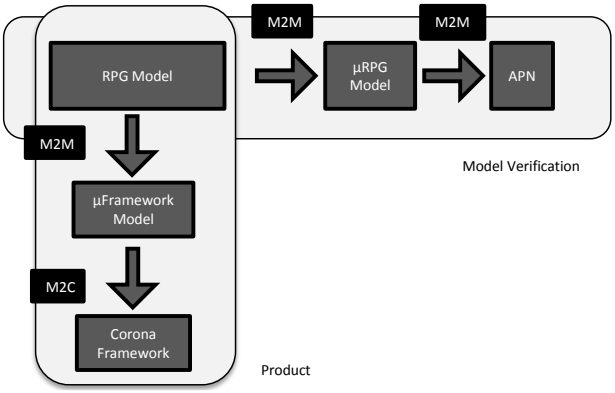


Figure 2. Transforming a source model to a target framework with model verification.

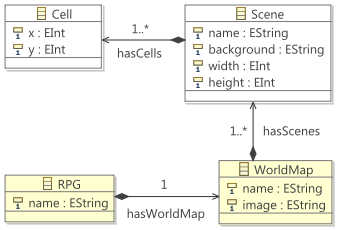


Figure 3. Excerpt of the RPG Meta-model.

design RPG games, and two other intermediate languages to simplify the process of transforming these instances to both the execution and analysis platforms.

In Figure 2, we can see the automated transformation specifications that are used to translate RPG game specifications into APNs, and to the code that is used in the Corona Framework. Instead of performing direct Model-to-Model(M2M) either from RPG models to APN, or Model-to-Code(M2C) from RPG models to the framework code, we decided to add intermediate steps to this process. The main reasons to introduce these intermediate steps are: *i)* to enable further reuse of the model transformation when approaching other target platforms (for both execution and analysis); *ii)* to enhance debugging capabilities by inspection on the results of the intermediate transformations; and *iii)* to ease and structure the implementation of the RPG language, and ease future language evolutions.

2.2.1 RPG Language Meta-model

The RPG Language Meta-model describes every possible feature of the defined RPG Domain. We used the feature model, created in the previous section, to design this Meta-model. It was annotated with OCL rules to guarantee that every produced RPG model is consistent by verifying rules such as "there can be only one hero in the world". This is crucial since both the execution and analysis transformations are assuming that the source RPG models are always correct.

In Figure 3, we show the part of the Meta-model that describes the Space entities. In this example, a RPG Game has only one World Map, that has a set of Scenes, which are composed by Cells. Cells are identified by x and y coordinates, a Scene is identified by its name, has an image for the background and its number of Cells is delimited by width and height. With this model of our RPG language, we generated a graphical editor to allow the Game

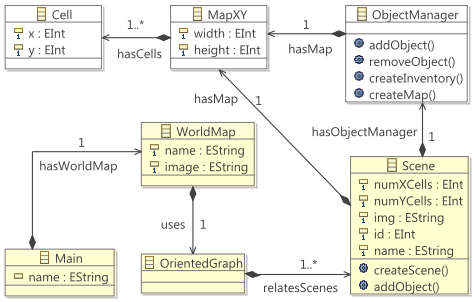


Figure 4. Excerpt of the μFramework Meta-model.

Designers to create RPG instances with it, as described in section 2.3.1. The created instances are then transformed into the other two languages: the Corona's Framework code, and the APNs used in ALPiNA.

2.2.2 μFramework Meta-model

The application domain of the chosen target framework (Corona) is very broad and general—thus it had only low-level primitives to draw and create graphical objects. Therefore, we built an API over it, in order to create the entities we needed to our RPG games and created a Meta-model for this API. This was called the μFramework API, which simplifies the M2C transformation to generate the code of the game. This way, we restricted the target framework to the RPG game context. The created Meta-model simply describes the functionality implemented in the μFramework API, mapping the entities to the functionality of the API.

In Figure 4, we can see that similarly to the RPG meta-model, this μFramework Meta-model has only one *WorldMap*, but now also uses a *OrientedGraph* that relates and store all the *Scenes* that the specified game may have. Each *Scene* has one MapXY, which holds all of the Cells of that Scene, and one *ObjectManager* that is responsible for managing the objects in the scene (e.g., placing an object, remove an object, etc.).

2.2.3 μRPG Meta-model

Given the analysis limitations of the ALPiNA model checker, we built another intermediate language (μRPG Language) which works as a filter that will only contains the entities we considered essential to check the properties related to the termination and score of a game. In this process we made several assumptions to reduce the number of entities to a minimum. As for example of one of these assumptions is that a hero can always beat an enemy, this has a huge impact since all enemies will be discarted from the process of model checking, which can lead to false positives where there may exist overpowered enemies that will block our way to the final goal.

The μRPG Meta-model one of modifications that occured in this meta-model is the concept of Partition which represents sets of adjacent cells that can be directly accessed by a hero: a given Partition holds the relevant information of that area w.r.t. the property of game termination—i.e., the objectives, the keys that can be picked up, the doors, and the hero position.

2.3 Implementation

In this section, we describe how we can create RPG models and use those models to automatically generate both the game source code and the models for verification.

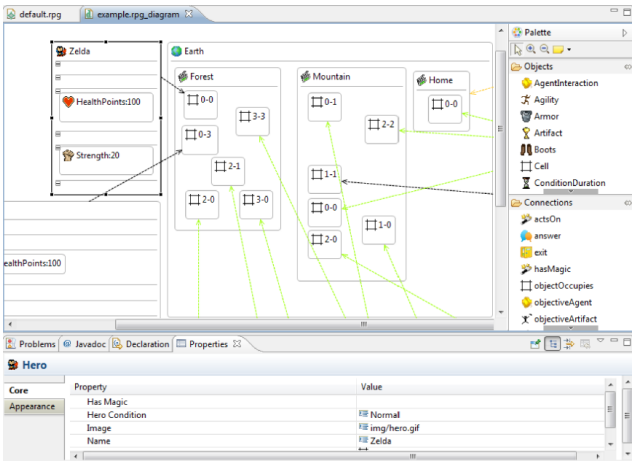


Figure 5. Screenshot of the graphical editor.

2.3.1 Graphical Language

We developed a graphical language based on GMF/EuGENia[4] that allows the creation of RPG models. The original RPG Language Meta-model was annotated with rules that describe how the entities and relations are represented in the GMF Editor.

In the Figure 5, we show an example of an RPG game created in this language. In the main window, the entities of the RPG model and the relations between them. In the properties window bellow, we can assign values to the RPG entities' attributes: in this case, we selected the Hero entity named 'Zelda'.

2.3.2 Generating the source code

The generation of source code for the RPG Game is divided in two transformation phases: First, we take a RPG Game instance and transform it to the μFramework Meta-model instance, using a M2M transformation. Then, we take the μFramework Meta-model instance and apply a M2C transformation, as can be seen in Figure 2.

All of the M2M transformations were done by means of the Atlas Transformation Language (ATL) [5], which is a textual model transformation language that is able to perform M2M transformations. The M2C transformation was specified using XPand[6], which is a template based tool to generate the source code of programs. Template based tools generate the source by reusing snippets of code. Those snippets are filled with values from the model instances. The information from the Meta-model is read in a visitor-pattern style, and it is used to generate the appropriate textual code. This technique is appropriate for cases where we already have a considerable amount of code from the target platform, and we want to reuse it in the MDD cycle.

2.3.3 Generating μRPG

To generate the μRPG Language, we made a simple M2M transformation that just propagates the relevant the entities from the RPG model to the model expressed in the μRPG Language. We implemented a breadth search algorithm to translate reachable adjacent cells in a scene into a Partition.

We then take the generated RPG model expressed in the μRPG language, and translate it again into an APN model to be analyzed

[4] http://www.eclipse.org/gmt/epsilon/doc/eugenia/

[5] www.eclipse.org/atl/

[6] http://wiki.eclipse.org/Xpand

in ALPiNA. This M2M transformation generates an APN from an μRPG model, where: *i)* each partition is mapped into a place; *ii)* Keys, objectives and the hero are mapped to tokens; *iii)* doors are mapped to transitions. In the translated APN there will always be two extra places: the KeySet that holds the keys picked up by the hero; and the Conquests that holds finished objectives. The transition associated with the door, allows the token associated with the hero to move from place to place, and will be enabled if and only if there is a token associated with the respective key in the KeySet place. We can put this token in the KeySet whenever the token associated with the hero is in the place that has that token. This also is applied to objectives.

Finally, to check if the game ends, we use AlPiNa to verify that there is a state which has the token associated with the final objective in the place Conquests. For the maximum score property, where we check if the cardinality of place Conquests can eventually be equal to the total number of objectives defined in the game.

3. Related Work

MDD is often applied to deliver a good separation of the concepts of the application from the concepts of the system. This separation improves productivity and communication because teams concerned with the domain of the application can easily talk with system developers without concerning the algorithmic requirements.

Besides the BATIC3S [15] project, there exists not so many evidence of successful application of MDD solutions in the SLE of a DSML. Nevertheless, related MDD approaches already have been introduced to the Game Industry before. In [14] it is proposed to adopt MDD in the Game Industry by proposing a modelling semi-automatic approach based on UML (that can be seen as a General Purpose Modelling language for Software Engineering) at several abstraction layers: Platform-Independent Models (PIM), Platform-Specific Models (PSM) and code level. However not demonstrated, the authors claim increasing productivity and higher code re-utilization, not mentioning how it might affect the error proneness caused by the fact the game developer must model and code, and keep track of consistency, in all the three layers. Code generation, in this case is not complete and can be seen more like a code skeleton generator.

There are several DSMLs built for game development. For instance, in [7] and [17] it was presented DSMLs for Adventure Games, with automatic code generation for a specific adventure game. However, our solution provides model checking capabilities (reducing costs with exhaustive game testing) and an abstract representation of the deployment platforms (tackling the problem of platform heterogeneity).

4. Conclusions and Future Work

In this work, we developed a DSML to create RPG Games with a complete MDD approach. This includes the M2M transformations and intermediate languages to tackle the complexity of building a DSML and the use of a model checker to verify some game properties.

The creation of an API over the framework allowed an M2M transformation that was easily produced between RPG and μFramework meta-models, which consists mostly of 1 to 1 relationships. This strategy of bottom-up modelling in the framework allowed the focusing on the RPG entities and the restriction of the power of the framework which was very low-level. The mapping of the RPG meta-model to APN is too much complex to simply perform it in just one step, therefore we created an intermediate meta-model μRPG that allowed a simpler mapping between both. We believe that the use of intermediate languages really help in this process, since we do not have to map complex entities directly to an APN. To complete the MDD cycle, the use of ALPiNA demonstrates that we can use a model checker to analyse and validate properties on RPG games, giving us a certain level of confidence about its implementation, since it passed verification phase.

Regarding the RPG metamodel evolution, the addition of a new feature should not affect the existing ones if its concept does not interfere with existing features. However if it does interfere, we have to analyze the impact of that interference in the μRPG Metamodel, which may lead to a partial redefinition of these model.

The choose of the DSL format (textual or graphical) has impact in the process of game development. A textual DSL may lead to a more readable solution than a graphical one in the development of big games, since a visual one will have problems displaying all the information about the game. However a graphical DSL allows the developer to get a preview how the things will be mapped, allowing the game developer detect errors faster.

As future work we are interested in collecting some metrics and conducting a complete assessment to gather the opinions from both game engine developers and game designers about their experience with this, or similar MDD approach. We are interested in comparing the productivity of game designers using this methodology and others. In a different survey, we will investigate the difficulty of developing a game language to generate games, in a MDD fashion, against the development of a generalist game engine, as the ones broadly used in the industry. Either for small game devices, such as the ones used in smartphones, or other more complex environments.

Acknowledgments

The presented work has been developed in the context of the following research institution: CITI fund PEst-OE/EEI/UI0527/2011 Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012, and the doctoral grant ref. SFRH/BD/38123/2007.

References

[1] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Verlag, 1st edition, 1999.

[2] J. Blow. Game development: Harder than you think. *Queue*, 1:28–37, February 2004.

[3] B. Boehm. Managing software productivity and reuse. *Computer*, 32(9):111 –113, sep 1999.

[4] D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. Alpina: A symbolic model checker. In *Petri Nets*, pages 287–296, 2010.

[5] D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. Alpina: An algebraic petri net analyzer. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 349–352. Springer, 2010.

[6] G. Caldiera and V. Basili. Identifying and qualifying reusable software components. *Computer*, 24(2):61 –70, feb 1991.

[7] A. W. B. Furtado and A. L. M. Santos. Using domain-specific modeling towards computer games development industrialization. In *Domain-Specific Modeling workshop at OOPSLA 2006*, 2006.

[8] O. M. Group. *Object Constraint Language OMG Available Specification Version 2.0*, 2006.

[9] G. Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[10] A. Kleppe. The field of software language engineering. In *SLE*, pages 1–7, 2008.

[11] C. Lakos. From coloured petri nets to object petri nets proceedings of 15th international conference on the application and theory of petri nets. 1995.

[12] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-*

Based Program Synthesis and Transformation. Proceedings of LOPSTR'99, LNCS 1817, LNCS 1817, pages 63–82. Springer-Verlag, Berlin, September 1999.

[13] M. Leuschel and T. Massart. Logic programming and partial deduction for the verification of reactive systems: An experimental evaluation. Technical report, University of Birmingham, 2002.

[14] E. M. Reyno and J. . C. Cubel. Automatic prototyping in model-driven game development. *Computers in Entertainment*, 7(2), 2009.

[15] M. Risoldi, V. Amaral, B. Barroca, K. Bazargan, D. Buchs, F. Cretton, G. Falquet, A. L. Calvé, S. Malandain, and P. Zoss. A language and a methodology for prototyping user interfaces for control systems. In *Human Machine Interaction*, pages 221–248. 2009.

[16] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[17] R. Walter and M. Masuch. How to integrate domain-specific languages into the game development process. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 42:1–42:8, New York, NY, USA, 2011. ACM.

[18] W. White, C. Koch, J. Gehrke, and A. Demers. Better scripts, better games. *Queue*, 6:18–25, November 2008.

ANNEX III

Case study: FlowSL

In this Annex we include the article [23], named **'Introducing Usability Concerns Early in the DSL Development Cycle : FlowSL Experience Report'**, which was published in Proceedings of the Model-Driven Development Processes and Practices (MD2P2) Workshop at MODELS conference in 2014.

FlowSL (Section 9.3) is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organization PSI and is integrated into MVC platform[1]. Work was developed under the collaboration of the Engineering Faculty of Porto (FEUP) and PSI. We applied action research to the development of a industrial oriented DSL FlowSL, in order to validate our solution design proposed in Section 4.2 as a part of our research process (Figure 1.1). Final report of this project can be downloaded from public repository [2].

[1] https://movercado.wordpress.com/ (accessed September 19, 2017)
[2] goo.gl/gQzX7B

Introducing Usability Concerns Early in the DSL Development Cycle: FlowSL Experience Report

Ankica Barišić[1], Vasco Amaral[1], Miguel Goulão[1], and Ademar Aguiar[2]

[1] CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Campus de Caparica, 2829-516 Caparica, Portugal
[2] Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
a.barisic@campus.fct.unl.pt, vma@fct.unl.pt, mgoul@fct.unl.pt, ademar.aguiar@fe.up.pt

Abstract. Domain-Specific Languages (DSLs) developers aim to narrow the gap between the level of abstraction used by domain users and the one provided by the DSL, in order to help taming the increased complexity of computer systems and real-world problems. The *quality in use* of a DSL is essential for its successful adoption. We illustrate how a usability evaluation process can be weaved into the development process of a concrete DSL - FlowSL - used for specifying humanitarian campaign processes lead by an international Non-Governmental Organization. FlowSL is being developed following an agile process using Model-Driven Development (MDD) tools, to cope with vague and poorly understood requirements in the beginning of the development process.

Keywords: Domain-Specific Languages, Usability Evaluation, Agile Development, Language Evaluation, Software Language Engineering

1 Introduction

Domain-Specific Languages (DSLs) and Models (DSMs) are used to raise the level of abstraction, while at the same time narrowing down the design space [1]. This shift of developers' focus to using domain abstractions, rather than general purpose abstractions closer to the computation world, is said to bring important productivity gains when compared to software development using general purpose languages (GPLs) [2]. As developers no longer need to make error-prone mappings from domain concepts to computation concepts, they can understand, validate, and modify the produced software, by adapting the domain-specific specifications [3]. This approach relies on the existence of appropriate DSLs, which have to be built for each particular domain. Building such languages is usually a key challenge for software language engineers. Although the phases of a typical DSL life cycle have been systematically discussed (e.g. [4, 5]), a crucial step is often kept implicit: the language evaluation.

DSLs are usually built by language developers in cooperation with domain *experts* [6]. In practice the DSL will be used by domain *users*. These domain *users*

are the real target audience for the DSL. Although domain *users* are familiar with the domain, they are not necessarily as experienced as the domain *experts* helping in the language definition. Neglecting domain *users* in the development process may lead to a DSL they are not really able to work with.

In this paper we apply action research to the development of a DSL, named FlowSL, designed to support managers in specifying and controlling the business processes supporting humanitarian campaigns. FlowSL is targeted to non-programmers. Their ability to use this language was identified as one of the highest concerns, so discovering usability issues in early development iterations, facilitated the achievement of an acceptable usability, while tracking the design decisions and their impact.

Usability has two complementary roles in design: as an *attribute that must be designed into the product*, and as the *highest level quality objective* which should be the overall objective of design [7].

This paper is organized as follows: Section 2 discusses related work; Section 3 provides a description of the evaluation approach; Section 4 discusses the language and evaluation goals and its development and evaluation plan; Section 5 discusses the lessons learned from the application of the described approach; finally, Section 6 concludes by highlighting lessons learnt and future work.

2 Related work

The need for assessing the impact of introducing a DSL in a development process has been discussed in the literature, often with a focus on the business value that DSL can bring (see, e.g. [8]). This business value often translates into productivity gains resulting from improved efficiency and accuracy in using a DSL [6], when compared to using a general-purpose baseline solution [9]. The quality in use of a DSL is, therefore, extremely important. In general, these assessments are performed with a final version of a DSL, when potential problems with the DSL are expensive to fix. A key difference in the work described in this paper is that we introduce language evaluation *early* in the DSL development process, so that problems can be found 'on-time' and fixed at a fraction of the cost it would take to fix them, if detected only in the deployment phase.

The term *quality in use* is often referred to more simply as *usability* [7], and includes dimensions such as *efficiency*, *effectiveness*, *satisfaction*, *context coverage* and *freedom of risk* (ISO 25010 2011). Usability evaluation investments have brought an interesting return on investment in software development [10]. Usability evaluation benefits span from a reduction of development and maintenance costs, to increased revenues brought by an improved effectiveness and efficiency by the product users [11].

Two important issues are *how* and *when* to assess DSL usability.

Concerning the *how*, we have argued that we can think of DSLs and their supporting editors as communication interfaces between DSL users and a computing platform, making DSL usability evaluation a special case of evaluating User Interfaces (UIs) [12] . This implies identifying the key quality criteria from

the perspective of the most relevant stakeholders, in order to instantiate an evaluation model for that particular DSL [13, 14]. These criteria are the evaluation goals, for which a set of relevant quantitative and qualitative measurements must be identified and collected. We borrow from UI evaluation several practices, including obtaining these measurements by observing, or interviewing, users [15]. In general, it is crucial that the evaluation of human-computer interactions includes real users [16], for the sake of its validity. In the context of DSLs, the "real users" are the domain users.

Concerning the *when*, we argued that we should adopt a systematic approach to obtain a timely frequent usability feedback, while developing the DSL, to better monitor its impact [17]. This implies the integration of two complementary processes: language development and evaluation. Software language engineers should be aware of usability concerns during language development, in order to minimize rework caused by unforeseen DSL usability shortcomings. In turn, usability designers should have enough understanding of the DSMs involved in software language development to be able to properly design the evaluation sessions, gather, interpret, and synthesize meaningful results that can help language developers improving the DSL in a timely way. This requirement is in line with agile practices, making them a good fit for this combined DSL building (i.e.software development) and evaluation process (i.e. usability design) [18].

3 Building usability into a DSL development process

Building a DSL may have a rather exploratory nature, with respect to the DSL requirements, particularly when the DSL is aimed for users with limited computational skills or poorly understood, or evolving domains. To build up a cost-effective and high quality process, we defined an agile and user centered DSL evaluation process [17, 13].

By placing DSL users as a focal point of DSLs' design and conception, the goal was to ensure that the language satisfies the user expectations. Besides involving Domain Experts and Language Engineers, as typically happens in the development of a DSL, we add the role of the Usability Engineer to the development team. Usability engineers are professionals skilled in assessing and making usability recommendations upon a given product (in this case, the DSL) and gathering unbiased systematic feedback from stakeholders [18].

Each development iteration focuses on a different increment or level of abstraction to be evaluated or refined. In the early phases it is important to study existing guidelines or standards for a particular domain and interview current or potential users about their current system or tools they are using to help them in accomplishing their tasks. This *context of use* study of a particular situation is intended to elicit the strengths and weaknesses of the baseline approach as well as the user expectations for the DSL.

Finally, once the language is deployed to users, an evaluation of its use in real contexts should be conducted, reusing the methods and metrics that were validated in the previous iterations.

4 Flow Specification Language (FlowSL)

The generic process described in the previous section was instantiated to the development of a concrete DSL — the FlowSL. FlowSL is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organization. FlowSL is integrated in MOVERCADO[3] (MVC), a mobile-based messaging platform at the core of an ecosystem that enables real-time and a more efficient impact, by facilitating interactions among beneficiaries, health workers and facilities, e-money and mobile operators. The platform is meant to allow data mining in the search of insights that can be used to improve the effects of the campaigns while supporting a high degree of transparency and accountability.

A first version of the system (MVC1) was developed as a proof-of-concept to validate the key underlying principles. The second version of the system (MVC2) was developed in the form of a platform easily customizable by managers and extensible by developers of the organization's team. An important goal was to develop a language, FlowSL, to empower the Campaign Managers to define new kinds of campaign flows taking advantage of their domain knowledge.

Without FlowSL, managers needed to specify the flows orchestrating their campaigns exclusively by means of presentations and verbal explanations. The implementation and evolution of campaigns often resulted in rework and unexpected behavior, usually due to vague specifications, incorrect interpretations, and difficulties in validating the implementation, a phenomenon known as *impedance mismatch* [19]. Therefore, the primary goal was to evolve the system to enable new users to easily create new campaigns and underlying flows. FlowSL is expected to enable the organization to streamline the process of defining campaigns and their base workflows, namely participants, activities, interaction logic, and messages.

4.1 FlowSL development process

In order to balance the development effort with effective reusability (e.g. while envisioning new marketing solutions), MVC2 was developed in a fast-paced way, iteratively, along six two-weeks sprints, following an agile development process based on Scrum[4] and best practices of evolving reusable software systems [20]. In the process of FlowSL development, the Domain Experts were part of the Product Owners team, while the Language Engineers were part of the Scrum Team. The DSL evaluation process was guided by the FlowSL development stages, as different effort was estimated in each sprint for its development.

The problem analysis was performed by mutual interaction and brainstorming between Domain Experts and Language Engineers in each sprint planning. Usability Engineers, in this case the researchers, had the role of observing and guiding the analysis outputs, while preparing the evaluation plan, without being directly involved in the language specification. To better understand and define

[3] http://enter.movercado.org/ (accessed in July 19, 2014)
[4] http://www.scrum.org/ (accessed in July 18, 2014)

the problem, the required functionalities were described in terms of small user stories. Also, the new description of the user roles was introduced as the FlowSL is expected to change existing organizational workflows. To improve interaction between the development team and the users, all the produced results from the analysis were continuously documented in a wiki. As Scrum suggests, the project management was based on a product backlog maintained and shared on-line.

The relationship between the MVC system, FlowSL development, and relevant language users and expected workflow is presented in Fig.1. The original MVC1 system was developed in a GPL (Ruby). FlowSL was first developed as a Ruby-based internal DSL. This approach allowed an optimal use of resources while keeping the existing system running. The second phase of language development was intended to support the managers to design the campaign flow specifications by themselves, using simple and understandable visual language constructs. In the planned third phase (future work), the focus will be on evolving the language's editor to be collaborative and web-based. It will also be an opportunity to work on language's optimizations in the generation process.

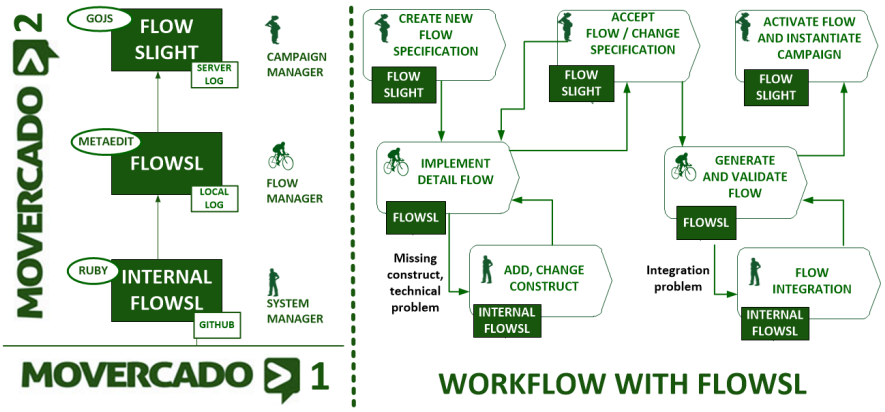


Fig. 1. FlowSL development and relevant language users with expected workflow

After defining the evaluation plan, the Usability Engineer prepared the usability requirements, using a goal-question-metric approach presented in Table 1, where goals conform to the Quality in Use model. These requirements were detailed and related to the right metrics and measurement instruments to perform appropriate usability tests in each development cycle. The validation of some of these requirements in earlier stages (e.g. understandability, readability) are stepping stones to achieve other soft requirements that cannot be evaluated in early phases (e.g. learnability). Multiple evaluations helped in validating and improving the set of identified metrics.

Table 1. Usability requirements

Requirement	*Metric*
Understandability: Does the user understand the different concepts and relations, and when and why to use each one of the concepts?	NCon - number of concepts, NRel - number of relationships NErrSpec - incorrect verbal definitions of total NCon and Nrel given in language NErrMod - incorrect interpretations of presented NCon and NRel given in modeled solution
Readability: How accurately is the user able to read the specified flows and interpret their meaning?	NConInst - number of concept instances in the model (flow), NReInst - number of relationship instances in the model NErrInst - number of incorrect verbal interpretation of NConInst and NRelInst given in language
Efficiency: How much time is needed for a user to read existing or specify a new flow?	TModInst - time necessary to read existing model instance (flow) TModSpec - time necessary to implement a new model instance (flow)
Effectiveness: Is the user able to correctly implement a flow from a given high-level description of the required flow?	NErrModInst - number of misinterpretation while reading existing model instance (flow) NErrModSpec - number of errors while implementing new model instance (flow)
Learnability: How much time is needed for users to learn the FlowSL language?	TLearNov - training time necessary to learn novice users to use language TLearExp - training time necessary to learn domain experts to use language
Flexibility How long does it take to quickly change or correct existing flow specifications?	TModEvol - time necessary to evolve model instance (flow) TModCorr - time necessary to correct incorrect implementation of model instance (flow)
Reusability How often user reuse existing flow specifications?	NModReuse - number of reusing existing model instance (flow) NModEvol - number of evolving existing model instance (flow)
Expressiveness Is the user able to specify all parts of flow?	NErrCon - number of concept, or its property that user is missing to implement model instance (flow) NErrRel - number of relationships, or its appropriate role that user is missing to implement model instance (flow)
Freadom of Risk Is the user able to implement the specifications in a way that can lead to unexpected or unwanted system behavior?	NEconDem - number of occurrence of economic damage due to incorrect flow specification NSofCor - number of occurrence of software corruption due to incorrect flow generation to system (flow)
Satisfaction How much is the user satisfied with FlowSL?	ConfLevel - self rated confidence score in a Likert scale LikeLevel - self rated likability score in a Likert scale

5 FlowSL evaluation and lessons learned

5.1 First FlowSL iteration: bottom-up approach (MVC2.1)

The *language goal* of the first iteration was to find the differences and commonalities in the Ruby code relevant for visual FlowSL and then do a corresponding mapping into a graphical representation, which would define the first draft of the concrete visual syntax of FlowSL. This is considered as a way to describe appropriate activities step by step by mapping relevant fragments of extracted code to a visual representation and to identify repetitive patterns that represent

reusable code artifacts. The *evaluation goal* was to assess whether this representation would be good enough to enhance the *understandability* and *readability* of flows from the perspective of Campaign Managers. It was expected that with the flow abstraction, the Domain Experts could describe more concrete requirements for the visual flow concepts.

The *evaluation intervention* was conducted when all existing flows of the MVC1 system were migrated to MVC2. This was the moment when the stakeholders could more clearly express the language purpose by distinguishing campaign processes from the flows underneath. The intervention was followed by an interview conducted with one representative *subject*: the Domain Expert with the role of Campaign Manager that was involved in specifying flows using the MVC1 system and who was also involved in the MVC2 Scrum development assuming, in that case, the role of Product Owner.

The *evaluation document* was prepared by the Usability Engineer containing 4 tasks: *Task 1* and *Task 2* describing user scenarios by roles and a global organization scenario that evaluator was asked to clarify and improve by placing him in organization workflow; *Task 3* presenting alternative feature models of FlowSL that are reviewed and redefined with a goal of separating campaign instantiation data and improving a vague language definition; *Task 4* presenting campaign flow based on simple and complex level of specification of the flow example (*IPC Validation*) that was found to be the most representative to describe. This task used metrics from the GQM table, which showed that the considered solution is very hard to understand.

The two major threats to validity of this evaluation were that it was subjective and only one user surrogate was involved. However, as the intended solution was seen as a step that helped to understand and to model the domain better, the guided interview helped to redefine the technical concepts using domain terms. Evaluation resulted in a clearer plan for the next development cycles as well as clarifying usability requirements and appropriate tasks. The textual FlowSL makes explicit all relevant domain concepts, but also many extra more. considered more technical, The performed evaluation helped the DSL developers to adjust the level of abstraction to the needs of the DSL end users. The language at this phase, could be used by the System Managers (knowledgeable of the concepts of the baseline system), but not by Campaign Managers.

5.2 Second FlowSL iteration: top-down approach (MVC 2.2)

The *language goal* of this iteration was to develop a visual FlowSL prototype using the MetaEdit[5] language workbench, that was selected for its support to top-down development. The *evaluation's goal* was to assess whether both the campaign managers and novice system managers were able to validate the specified flows using the newly proposed visual language and editor. These evaluations covered also the effectiveness and expressiveness of the target language.

[5] http://www.metacase.com/ (accessed in July 19, 2014)

The *First evaluation intervention* was organized very quickly and involved interviewing two *subjects*: the campaign manager from the first development iteration and the system manager who was involved in the DSL development. The intervention consisted of one task where the subjects had the opportunity to compare two alternative concrete flow representations for the same ongoing example.

Based on the evaluation results the Usability Engineer produced designs of the concrete syntax for the DSL development team.

The *second evaluation intervention* involved the same subjects. The *evaluation document* had three tasks: Task 1 focused in assessing the *understandability* and *expressiveness* of the individual symbols; Tasks 2 and Task 3 meant to measure the *readability* and *efficiency* of the designed solution of the simple and complex flow. In addition to that, the Domain Expert was asked to describe the use of the symbols from Task 1 to produce the presented flow solutions and to describe the situations in which the existing flows can be reused. The evaluation session with the System Manager made it possible to identify important missing relationships between FlowSL concepts, as well as their connection points (hot spots) with the MVC system underneath.

For the *third evaluation intervention* the usability engineer introduced the design improvements motivated by the feedback obtained the previous evaluation. The new notations were designed and implemented, to be again compared. The tasks were similar to the previous intervention, although more elaborated. Here, the same subjects from the previous interventions were involved, as well as a member of the Scrum team.

For this third intervention the rules related to the usage of a certain activity were discussed. The usability engineer evaluated the cases where the system manager would have the need to hack the existing campaign flows, in order to customize certain functionality or rule. The goal was to use an example-based approach to identify improvements in the language.

It became clear that the evaluation materials prepared earlier helped to speed up the following evaluation phases and reduced their implementation costs. Besides, they became templates for the corresponding learning materials. Also, it was possible to abstract the language one level further, so that an online visual editor was built to support rapid high level specifications of flows. To better deal with the increasing complexity of the specified models, rather than presenting all the concepts related to the flow definition visually, a better option would be to present just high level concepts that are reused often, while others are hidden and based on predefined rules that can be eventually reconfigured textually. This approach empowered both the domain experts and the product owners to better control the design decisions.

6 Conclusions and future work

In this paper, we presented an experience report on how to integrate top-down usability engineering practices into a bottom-up agile development of a DSL

from its beginning. While playing the role of Usability Engineers, we experienced that small iterations involving Domain Experts, Product Owners and End Users can help us to clarify the meaning and the definition of the relevant language concepts. This enables an early identification of possible language usability shortcomings and helps reshaping the DSL accordingly.

Early evaluations can be executed with a relatively low cost thanks to model-driven tools that support production of rapid prototypes and presenting the idea. These evaluations support well-informed trade-offs among the strategy and design of the DSL under development, and its technical implementation, by improving communication. Besides, they improve the traceability of decisions, and of the solution progress. These iterations also help to capture and clarify contractual details of the most relevant language aspects that need to be considered during DSL development, and are a key element to improve the End Users experience while working with FlowSL.

We plan to validate our decisions, metrics, and the overall merit of the developed DSL, by performing experimental evaluations with both expert and novice users, by making comparisons to the baseline approach in Ruby, as well as to other process modelling languages that are natural candidates to serve for similar purposes (e.g. BPMN, JWL).

An additional step is to conceptualize the traceability model of design changes and evaluate its impact on the decision making process. We expect that in each iterative evaluation step we will not only identify opportunities to improve the usability of the DSL, but also to improve the evaluation process itself (e.g. through the validation, in this context, of the chosen metrics).

Weaving usability concerns into agile process is helping us to continuously evolve FlowSL, improving the cost-effectiveness of DSL usage in specifying campaigns, and supporting a clearer assessment of which language concepts are more relevant to the different kinds of language users, which in turn helps finding the right level of abstraction and granularity of concepts. All these benefits come with the cost of adding usability skills and of introducing new practices in the agile process, namely the introduction of lightweight metamodeling tools. The balance however, seems to be very positive, but ROI should be calculated precisely to support this claim.

References

1. Gray, J., Rossi, M., Tolvanen, J.P.: Preface. Journal of Visual Languages and Computing, Elsevier **15** (2004) 207–209
2. Kelly, S., Tolvanen, J.P.: Visual domain-specific modelling: benefits and experiences of using metacase tools. In Bézivin, J., Ernst, J., eds.: International Workshop on Model Engineering, at ECOOP'2000. (2000)
3. Deursen, A.V., Klint, P.: Little languages: Little maintenance? Journal of Software Maintenance: Research and Practice **10**(2) (1998) 75–92
4. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys **37**(4) (2005) 316–344

5. Visser, E.: WebDSL: A case study in domain-specific language engineering. In Generative and Transformational Techniques in Software Engineering II, Ralf Lämmel, Joost Visser, and João Saraiva (Eds.). Lecture Notes In Computer Science **5235** (2007)
6. Voelter, M., Dietrich, C., Engelmann, B., Helander, M., Kats, L., Visser, E., Wachsmuth: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform (2013)
7. Petrie, H., Bevan, N.: The evaluation of accessibility, usability and user experience. Human Factors and Ergonomics. CRC Press (2009)
8. Kelly, S., Tolvanen, J.P.: Domain-specific modeling: enabling full code generation. John Wiley & Sons (2008)
9. Kosar, T., Mernik, M., Carver, J.: Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical Software Engineering **17**(3) (2012) 276–304
10. Nielsen, J., Gilutz, S.: Usability return on investment. Technical report, Nielsen Norman Group (2003)
11. Marcus, A.: The ROI of usability. In Bias, Mayhew, eds.: Cost-Justifying Usability. North- Holland: Elsevier (2004)
12. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: Quality in use of domain-specific languages: a case study. In: Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. PLATEAU '11, New York, NY, USA, ACM (2011) 65–72
13. Barišić, A., Monteiro, P., Amaral, V., Goulão, M., Monteiro, M.: Patterns for evaluating usability of domain-specific languages. Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012 (October 2012)
14. Kahraman, G., Bilgen, S.: A framework for qualitative assessment of domain-specific languages. Software & Systems Modeling (2013) 1–22
15. Rubin, J., Chisnell, D.: Handbook of Usability Testing: How to plan, design and conduct effective tests. Wiley-India (2008)
16. Dix, A.: Human computer interaction. Pearson Education (2004)
17. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: How to reach a usable DSL? moving toward a systematic evaluation. Electronic Communications of the EASST **50** (2011)
18. Lárusdóttir, M., Cajander, Å., Gulliksen, J.: Informal feedback rather than performance measurements–user-centred evaluation in scrum projects. Behaviour & Information Technology (ahead-of-print) (2013) 1–18
19. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information systems **27**(6) (2002) 365–389
20. Roberts, D., Johnson, R.: Evolving frameworks: A pattern language for developing object-oriented frameworks. In: Proceedings of the Third Conference on Pattern Languages and Programming, Addison-Wesley (1996)

ANNEX IV

Case study: Visualino

In this Annex we include the article about evaluation of the DSL Visualino (Section 9.4), for the programming low-cost robots, under a title **'Leveraging teenagers feedback in the development of a Domain-Specific Language – the case of programming low-cost robots'**. This article is published in ACM SAC conference Action IC1404 MPM4CPS in 2018. The work was developed under the collaboration between the group ASE NOVA/LINCS and Artica[1], a company that specialises in the development of robotic and audio-visual solutions.

The first language design was developed in 2013 in the context of the master thesis [135], where language was named Farrusco. Later, language continue to evolve and was renamed to Visualino after second release in 2015, and recently Gyro, after a third release in 2016. We manage to show that the usability of the Gyro significantly improved in terms of efficiency and satisfaction when compared to earlier version. To consult the details, take a look at experiment repository[2].

This study present application of our usability evaluation approach (Section 4.2) on the industrial case study during several development iteration. As this, it served as design validation example of our research process (Figure 1.1). Further, it was used to validate feasibility of our approach (Chapter 5) as a USE-ME model instantiation example (Section 6.2) [3]. Further, it served as a running example for our integration study (Section 9.6, Annex VI) [4].

This work was supported by FCT/MEC NOVA LINCS; PEst UID/ CEC/04516/ 2013 and DSML4MA TUBITAK/0008/2014 Projects.

[1] http://artica.cc/ (accessed September 19, 2017)
[2] https://sites.google.com/view/vl-empiricalstudy/home (accessed September 19, 2017)
[3] https://github.com/akki55/useme/tree/master/examples/pt.fct.unl.novalincs.useme.example.Visualino
[4] https://github.com/akki55/useme/tree/master/examples/gyro

Leveraging Teenagers Feedback in the Development of a Domain-Specific Language

The Case of Programming Low-Cost Robots

Ankica Barišić, João Cambeiro
NOVA LINCS, DI, FCT
Lisboa, Portugal
a.barisic@campus.fct.unl.pt
jmc12976@campus.fct.unl.pt

Vasco Amaral, Miguel Goulão
NOVA LINCS, DI, FCT
Lisboa, Portugal
vma@fct.unl.pt
mgoul@fct.unl.pt

Tarquínio Mota
Artica Creative Computing
Caparica, Portugal
tarquinio@gmail.com

ABSTRACT

Domain Specific Languages (DSLs) empower end-users to express software tasks that were traditionally developed by software engineers. DSLs allow users to express themselves in terms closer to the way they think about their problems, rather than in computational terms. However, conceiving a DSL with an adequate user experience for its end-users is not a trivial task, and the process of engineering that adequacy tends to be performed *ad-hoc*. The Gyro Creator Language (GCL) is an open-source DSL for controlling low-cost rover-like Arduino robots, designed for being used by teenagers with no previous computing skills, so they can be introduced to programming in a fun way. In this paper, we discuss an iterative process building on teenagers' early feedback, collected in a series of empirical evaluations with 128 teenagers, and how this has helped us driving GCL to a competitive level in terms of usability, when compared to well-established alternatives such as Lego, or Scratch.

CCS CONCEPTS

• **Software and its engineering → Software usability**; **Domain specific languages**; **Visual languages**;

KEYWORDS

Programming languages for children, Robotics Programming

ACM Reference format:
Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2018. Leveraging Teenagers Feedback in the Development of a Domain-Specific Language. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 9 pages.
https://doi.org/10.1145/3167132.3167264

1 INTRODUCTION

The increasing software pervasiveness fosters a growing concern for making some of its development accessible to end-users with no formal training in programming. Creating Domain-Specific Languages (DSLs) for empowering end-users is challenging, as we need to bridge the gap between computation concepts and the concepts mastered by the end-users. Two complementary ways of bridging this gap are (i) nurturing 'computational thinking' [37] skills in end-users, and (ii) devising adequate metaphors that hide the unnecessary complexity of computational concepts from those end-users. We were contacted by a company interested in developing an open source web-based DSL, called Gyro Creator Language (*GCL*), formally known as *Visualino*, targeted to teenagers, to empower them to control low-cost rover-like Arduino robots. The challenge was how to assess the DSL in a timely way, so that end-user feedback could lead to a competitive product. In particular, the company was concerned with the user-friendliness of the DSL.

However, the development process of DSLs lacks a systematic and iterative approach to evaluating and detecting usability issues since the early stages of the DSL construction. Therefore, we add to the common iterative life cycle of DSL development an evaluation task involving the end-users, to be performed in each iteration. The teenagers' feedback is used to help to steer the GCL's evolution through the identification of several improvement opportunities in the language. This evaluation stage in each iteration is often not reported in the context of developing DSLs but is key for our development effort. In this assessment, we contrast GCL with two popular DSLs that are used to control rover-like robots: a commercial competitor (Lego)[22] and an open source initiative (Scratch)[32].

We report on the design and results of the empirical studies used in this evaluation that helped us identifying the language's strengths and weaknesses. We discuss how this lead to the improvement of the *GCL* language. To help to achieve this higher level goal, we answer two more detailed research questions:

- ***RQ1:*** *How does the current GCL (GCL2) compare to baselines (a previous version of GCL (GCL1), Lego and Scratch) regarding the* ***Effectiveness*** *of the teenagers when programming a robot?*
- ***RQ2:*** *How does the current GCL (GCL2) compare to baselines (a previous version of GCL (GCL1), Lego and Scratch) regarding the* ***Satisfaction*** *of the teenagers when programming a robot?*

The chosen baselines are aimed at providing a comparison basis with (1) a previous version of GCL (GCL1), so that we can assess the extent to which the feedback collected with that previous version has helped in its evolution and (2) two popular competitor languages, so that we can assess how competitive the GCL can be, when contrasted with those languages.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167264

2 RELATED WORK

2.1 Language Usability

Language usability is the degree to which a language can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency and satisfaction in a specific context of use (adapted for the particular case of languages from [13]).

User-centered design (UCD) [28, 36] can contribute to more usable DSLs. For example, [1] presented an innovative visualisation environment, which eases and makes more effective the experimental evaluation process, implemented with the help of UCD. A visual query system was also designed and implemented following the UCD approach [6]. Although there is a lack of general guidelines and best practices to conduct language usability evaluations, they are slowly being recognised as an important step in the Language Engineering life-cycle [20]. An iterative approach allows us to trace usability requirements and the impact of usability recommendations througout the DSL development process [3].

2.2 User characterization and evaluation

Teenagers who are familiar with computers and technologies tend to be more successful in new computer-related tasks [12]. Also, they are likely to work and play in groups, (e.g. sharing a single computer) [8]. Involving teenagers as subjects in empirical studies is a valid option to evaluate usability on software products targeted to teenagers [34] but also challenging, as teenagers have a high variability in cognitive and development abilities at a given age [7]. Nevertheless, it has been shown that teenagers can identify and report usability problems using methods like 'think aloud', interviews or questionnaires [25].

Previous studies (e.g. [15, 18, 31]) concentrate on issues related to K12 courses curricula (i.e. covering from kindergarten to 12 years of basic education), the motivation of students to engineering, and education of computational teaching. Teenagers have been used in the past as subjects for language-related studies. For example, a hybrid approach combining text and visual notations offers the best compromise to increase efficiency and effectiveness of teenagers while using that language [19]. However, we are not aware of other formal studies concerned about improving software language usability, involving teenagers as subjects.

2.3 Programming languages for teenagers

Current robot platforms offer Application Programming Interfaces (APIs) for programming with languages such as *Java* (e.g. [9]), [30]) or *.NET Framework* languages (e.g.[14]). These languages may exclude many teenagers who are not (yet) proficient in programming. Without proper training, it is hard for them to program and master a textual programming language with a complex syntax full of technical concepts. This is true even with the help of powerful Integrated Development Environments (IDEs). We argue it is far from trivial to design a DSL for teenagers (and even more so for younger children).

There are several programming languages designed specifically for children (including teenagers). Examples include *Alice* [10], *Blockly* [11], *Lego* [22] and *Scratch* [32]. We used for comparison purposes with GCL *Lego*, one of the most widely used languages for programming toy robots, and *Scratch*, a popular visual programming language for teaching programming concepts to children which can also be used for controlling rover-like robots.

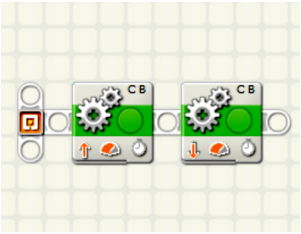


Figure 1: LegoMS

Figure 2: Scratch

2.3.1 Lego. Inspired by Papert's seminal work [29], *Lego Mindstorms (LegoMS)* is one of the most well-known technologies, in educational robotics for children. Several case studies have been done for this technology in order to understand which are the main individual needs of the children, when working with robots. *LegoMS*'s technology combines hardware and software, so that users can develop and deploy programs specifying behavior to be executed by the *Lego* robot.

Lego Mindstorms NXT 2.0, used in our usability trials, presents building blocks (like bricks) as elements to build a program. Each block represents a programming concept, such as an execution control element, *e.g.* loops, conditions, arithmetic, or an actuate block that interacts with the robot components, *e.g.* motors. The sequence of blocks is constructed by a behavior flowchart, structuring the program's blocks.

LegoMS has an appealing notation and is used only with relatively expensive *Lego* robots. However, the development of complex behaviors may become difficult [26]. The increasing number of blocks and the size of the diagram make it difficult to analyze and read the program solution. Non-trivial behaviors are difficult to implement in a visual programming language like this. Fig. 1 exemplifies the move back and forth example. The diagram contains two composition elements describing how the robot should move.

2.3.2 Scratch. Scratch [32] relies on actions (Blocks) to operate specific objects (Sprites). They can be seen as a visual abstraction to the Object Oriented programing paradigm with some restrictions (no support for custom objects and the dynamic generation of sprites). Open source tools like *mBlock* [33] and *Enchanting* [21], are built upon *Scratch*, and meant to be simple and used for robot programming. *mBlock* is a solution which compiles to the *Arduino* open source hardware platform, which makes it suitable for our assessment study, as it uses the same hardware as GCL.

Fig. 2 illustrates how to program the robot to move forward during one second and to go backwards during one second. However, as this language is general purpose, it does not have abstractions of operations like move forward or backwards. Therefore, the user needs to detail the movement operator. This includes to describe the pins of each motor on the Arduino board and deal with concepts like motor rotation directions, rotation speed, angular velocity.

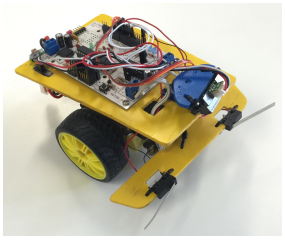


Figure 3: Arduino Robot

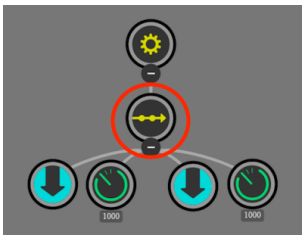


Figure 4: GCL1 - Back and Foward

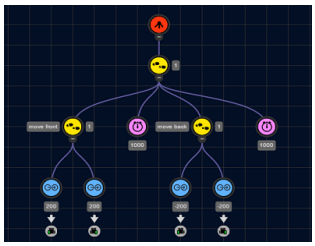


Figure 5: GCL2 - Back and Foward with bumpers

3 CONTEXT

GCL is a visual language that allows the user to implement programs for robot behavior, through the manipulation of visual elements, or objects. This manipulation is expected to help the user to understand quickly the programming mechanisms. Visual languages are thought useful for introducing programming concepts to children [16], while robots perform the developed code in the real world. The observation of the program running on a physical robot provides an engaging feedback on the implications of changes in the program.

The robot is an *Arduino* board with a set of sensors and actuators. *Arduino* is an open-source prototyping platform [2]. The low-cost robot (see Fig. 3) which works with GCL has the following configuration: (i) An infra-red distance sensor; (ii) Two bumpers (e.g. collision detectors sensors); (iii) One motor for each wheel; (iv) A Servo motor that actuates under the distance sensor; (v) A simple LED light.

Programming the *Arduino* textual code requires technical skills not owned by most teenagers. To mitigate this problem, the robot may be programmed using *GCL*, which is then automatically translated into the *Arduino* textual code.

GCL's visual syntax is based on the behavior tree paradigm [24], a mathematical model of plan execution, used for a diversity of areas including robotics, control systems and software games. A complex behavior is mapped into smaller and simpler behaviors through its branches. This descending order of complexity provides a structured way of defining complex behaviors (which are used to define objectives) through simple tasks defined hierarchically. Each node may have a specification that determines how the actions of its children will be executed (in parallel, or sequentially). The child node returns its status to the parent node, and this successively happens until the root of the tree.

Fig. 4 illustrates how to program the robot to move back and forth. The root node is decomposed into a single sequential node (highlighted with a red circle). The sequential node runs all its children (in this case, the leaf nodes) in depth first traversal order. Leaf nodes represent the most primitive actions that could be taken by an agent. In Fig. 4 the nodes represented by arrows pointing downwards are outputs actions. In this case they are left and right motors commands. The outputs can also be visual using LEDs or audio using a buzzer. Both nodes displayed by 2 round timer shapes are wait commands in which the execution of the next node is delayed by the duration determined by the end user for each node.

3.0.1 From GCL0 to GCL1. The first development iteration of *GCL* GCL0 started with a domain analysis, followed by the design, implementation, and evaluation, in order to quickly deploy an early prototype. The last phase involved 22 (10+12) children, with the age range from 8 to 12, as subjects included in an exploratory study [23]. Children under 10 had a hard time learning *GCL*. Further development resulted in the next release GCL1 (Fig. 4), which improved the interaction model and provided a web based solution.

3.0.2 From GCL1 to GCL2. GCL1 was evaluated experimentally and compared to the one of the most popular commercial competitors in the market, *LegoMS*. Based on the results of this empirical study, some improvements were suggested and applied to GCL. The focus was on improving the user interface providing better readability of the programs being developed and improving error prevention. The suggestions were:

- Improve error feedback and suggestions to solve problems.
- Add auto-alignment of new nodes added to a sequence node, preventing situations where the visual order of nodes from left to right did not match the order of their execution.
- Highlight when a user selects a new node to add to the diagram the nodes that are available to form a connection. This provides better user assistance and error prevention.
- Allow the users to create blocks of tree structures. This feature promotes reuse of previously developed structures and allows to share and slowly introduce more complex behaviours to novice users.
- Show the Icon's label with mouse-over events. This helps the user to recognize the available options.
- Introduce a different method of collapsing and expanding tree structures. The use of the keys "+" and "-" for these operations was not clear to the user.
- Introduce new zooming options to allow an easier navigation of the tree structure by introducing a fit to screen operation of the entire tree structure or only the selected nodes.

These suggestions lead to the release of a new improved version of the language *GCL2* (Fig. 5). Finally, GCL2 was compared with Scratch.

4 EXPERIMENT PREPARATION

In this section, we describe the experiment protocol. Additional materials can be found in this paper's companion site [4]. We had two runs of the same experiment, with the second prototype release (GCL1) which we compared to Lego; and second GCL version

(GCL2) which was compared to Scratch. The experiment was announced as a robot programming challenge to engage teenagers to participate.

During the first run, GCL1 was compared to the best product in the area but appropriate only for costful Lego robots. Therefore, the robots used in this run were different. However, during the second experiment run, both GCL2 and Scratch programs were running on the same robot (see Figure 3). The two different experiment runs were designed as two parts of the same experiment. Both participant's profile and the tasks were similar, so that, in practice, we can think of them as a single experiment with four different languages (two of which are different versions of the same language).

4.1 Experiment objectives

The high-level objective of our study is to evaluate the usability of *GCL*, *"Usability of Programming the robot"*, as it is expected to be used by a broad group of people that are not expected to have previous programming skills.

As we already have different generations of a functional prototype of GCL, we measure *GCL*'s *effectiveness* and *satisfaction*. The metrics for these usability requirements are defined in Table 1. We are not particularly concerned with other requirements such as efficiency or learnability at this point because we are at a stage of the language development where we want to know if the teenagers can program the robot, which is a prerequisite for measuring other characteristics. The experiment objectives follow the GQM template [5] and are defined as follows:

G1 - Analyse the effect of *GCL2*, **for the purpose** of evaluation, **with respect to** its impact on the *effectiveness* in programming a robot **when compared to** three *baselines*, namely *GCL1*, *Lego* and *Scratch*, **from the point of view** of researchers, **in the context of** an experiment conducted with secondary school subjects.

G2 - Analyse the effect of *GCL2*, **for the purpose** of evaluation, **with respect to** its impact on the *satisfaction* in programming a robot **when compared to** three *baselines*, namely *GCL1*, *Lego* and *Scratch*, **from the point of view** of researchers, **in the context of** an experiment conducted with secondary school subjects.

Table 1: Usability requirements

Requirement	*Metric*
Effectiveness: Is the <User> able to correctly implement a given <Use Case>?	PCorrUCInst - percentage of correctly implemented concepts (i.e. garanteeing that an expected outcome is reached) in a given <Use Case>
Satisfaction How much is the user satisfied with *GCL*?	ConfLevel - self rated confidence score in a Likert scale LikeLevel - self rated likeability score in a Likert scale LearnLevel - self rated learnability score in a Likert scale

In particular we test the following (null) hypotheses:

- $H1_0$: Using *GCL2* has no influence in the *effectiveness* of programming the robot when compared to programming the robot with the *baselines GCL1*, *Lego* and *Scratch*.
- $H2_0$: Using *GCL2* has no influence in the *satisfaction* of programming the robot when compared to programming the robot with the*baselines GCL1*, *Lego* and *Scratch*.

4.2 Experiment design

We used a *between groups* design where participants were randomly assigned to the task of either programming a robot using *GCL1* or *Lego*, in the first run, or *GCL2* or *Scratch*, in the second run. Each participant only participated in one of the four alternatives to avoid learning effects. The experimental process was similar for all runs and subject groups, starting with a learning session that lasted 30 minutes, during which the participants filled in a background questionnaire and learned about programming robots using their assigned language. Then, the participants had 15 minutes to solve a programming challenge. The contents of the computer screen during the session were recorded. Finally, there was a feedback session, taking up to 5 minutes, to collect the teenager's subjective opinions about using the language they were assigned to.

4.2.1 Participants, teams and groups. The participants were high school students recruited through convenience sampling, among the visitors of two different open doors days in the same university. In each day, participants were randomly assigned to one of the two groups: in the first day, there was a group with GCL1 and another one with Lego; in the second day, there was a group with GCL2 and another one with Scratch. Students and their teachers were aware of this study and volunteered to participate. Two lecturers guided each language's groups. The teenagers were requested to participate, in teams of up to three elements, using their assigned language and corresponding robots. Each group had a maximum of 5 teams participating in the same session. Undergraduates helped in the experiment, while a researcher monitored the data collection process.

4.2.2 Technical, social and physical environment. Each team worked on a desktop computer with OS Windows 7 Professional (Athlon 64x2 Dual Core 5000 2.6 GHZ processor, 4 GB RAM and a 17-inch monitor with a screen resolution configured at 1280x720) to implement the applications. The interaction between user and computer was achieved by the use of keyboard, mouse, and screen, and was captured by *Debut* video recorder [27].

The atmosphere was set up to be challenging and educative, but also playful and entertaining, to keep the teenagers still interested to participate and to reduce the sensation of failure. Team members were allowed to talk to each other. They could seek help if they had technical issues with the robots.

4.2.3 Exercises and Challenge. During the learning sessions, participants solved three basic robot programming exercises. They tested each of those exercises on the corresponding robot. Furthermore, they were encouraged to ask questions and to ask for help if necessary.

The first exercise was to program the robot to *move forward* and then *move back* (see Fig. 6). It gave the participants a notion of how to program the robot to move in a basic way. The numbering in Figures 6 through 10 represents the order in which the sequential commands would be executed. The second exercise was to program the robot to move along a path similar to a '5' (see Fig. 8).

In each turn, the robot needed to execute the move operation using the same amount of time and then to make a 90^o angle turn. The third exercise was to program a robot to *move forward* until it *bumped* into some object, then it would *move back* and *stop* (see

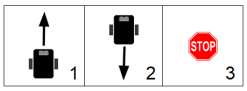


Figure 6: Exercise 1

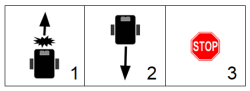


Figure 7: Exercise 3

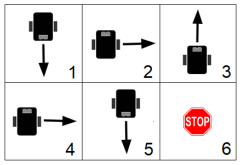


Figure 8: Exercise 2

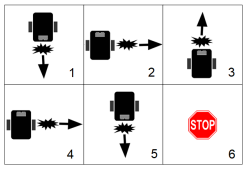


Figure 9: Challenge 1

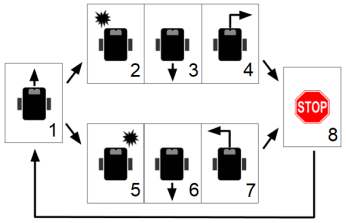


Figure 10: Challenge 2

Fig. 7). During this exercise, participants learned how to use the detector sensors and to compose their behaviour in a sequence with other modelling elements.

The challenge in the first experiment was to program the robot to make a shape of '5' but the robot would only turn when it would hit the bumper (see Fig. 9). It was composed by basic user story actions of the exercises. The challenge in the second experiment was slightly different (see Fig. 10). The robot was expected to detect if it hits the obstacles with the left or the right bumper, and it was supposed to turn accordingly to the opposite direction. It either executed the instructions associated with hitting the right bumper, or those associated with hitting the left bumper.

When the team thought they had a solution, they were invited to physically test it in the arena with a given robot. If they were not happy with the result of the test, they could go back and try to fix whatever was wrong with their solution.

4.3 Experimental instruments and measurements

During the experiment, we used survey forms, video recordings and a competition arena to collect data for further analysis. The survey forms were composed of *"Smileyometers"* which are found to be appropriate for teenagers questionnaires [35]. While answering to the forms, the teenagers were assisted by an adult (one of the experiment assistants), to ensure that there were no misinterpretations of questions and answers and to confirm that the participants did not experience reading problems. As we grouped the participants into teams, the participants' individual answers to questionnaires were merged. We computed the mean response within each group, for each answer.

Table 2: Instruments and Scales

	Instrument	*Value*
Profile	Background Questionnaire	0 / 0.5 / 1
Effectiveness	Video recording	0 / 1
Satisfaction	Satisfaction Questionnaire	-1 / 0 / 1

Profile is a measure influenced by *Experience* factors on Computer Games, Programming or Programming a robot, and *Tendency* factors reflecting the teenager's tendency to Mathematics, Physics or to Learn programming. The data for calculating the *Profile* was collected through the background questionnaire, that consisted of questions designed to assess those pre-defined factors. Each answer was encoded in three possible answers: 'Yes' (with a score of 1), 'Intermediate' (0.5), and 'No' (0).

We used the recorded videos to evaluate the **Effectiveness**. The challenge could be solved by composing elements of the training exercises, which are marked as Success (S) or Failure (F). Effectiveness measures the percentage of modelling elements correctly built and composed to achieve the solution.

Satisfaction is characterized by: a *Confidence* factor, reflecting ConfLevel metric that tells how confident teenagers were about their solution; *Likeability* factor, reflecting LikeLevel metric that tells how interesting and enjoyable they found the challenge itself; and, *Learnability* factor, reflecting LearnLevel metric that tells how useful they found what was taught during the learning session which helped them to face the final challenge. The data was collected trough a satisfaction questionnaire, that consisted of questions designed to assess the defined factors. A 'Yes' scored 1 point, 'Intermediate' scored 0, and 'No' scored -1.

5 RESULTS

5.1 Descriptive statistics

Table 3 presents the descriptive statistics for the metrics collected in our data analysis. In the *Characteristic* column we present the property under scrutiny. For each of these characteristics, we show four rows, one for each *Language* (in the second column). We further detail the *mean*, *standard deviation*, *skewness*, *kurtosis*, and the *p-value* for the Shapiro-Wilk normality test. The number of participant teams is not always the same for all languages. The shape of the distributions concerning most of the variables suggests that, in general, normality is not a reasonable assumption (p-value < 0.05). The exceptions are metrics concerning *profile*, *experience* and *tendency*, where, for most languages, normality is a reasonable assumption (*p-value* >= 0.05). The visual inspection of boxplot diagrams, Q-Q plots and kernel density plots (omitted here for the sake of brevity) further reinforced our assessment concerning data normality. As several of the variables have a non-normal distribution, we assume this for the remainder of our analysis. Note also that all teams expressed the maximum *confidence* level on their solution for GCL2, as well as the maximum *likeability* level for GCL2 and Scratch.

The variance of the distributions is not similar, when comparing the characteristics metrics distributions for the different languages, as shown in table 4.

Table 3: Descriptive statistics

Characteristic	Language	N	Mean	Std. Dev.	S-W
Age	Scratch	8	16.7062	1.23782	0.023
	GCL2	9	16.3689	0.91408	0.180
	GCL1	17	16.6059	1.31694	0.018
	Lego	14	16.4429	1.28465	0.039
Profile	Scratch	8	0.6875	0.17107	0.542
	GCL2	9	0.6019	0.13029	0.213
	GCL1	17	0.6225	0.16435	0.455
	Lego	14	0.6250	0.22349	0.201
Experience	Scratch	8	0.5625	0.23465	0.241
	GCL2	9	0.4259	0.14699	0.338
	GCL1	17	0.5294	0.24463	0.214
	Lego	14	0.4643	0.31473	0.042
Tendency	Scratch	8	0.8125	0.18767	0.197
	GCL2	9	0.7778	0.20412	0.122
	GCL1	17	0.7157	0.20211	0.059
	Lego	14	0.7857	0.2210	0.004
Effectiveness	Scratch	8	0.8000	0.23905	0.041
	GCL2	9	0.9556	0.08819	0.000
	GCL1	17	0.3765	0.40548	0.001
	Lego	14	0.7857	0.32783	0.000
Satisfaction	Scratch	8	0.9583	0.05893	0.002
	GCL2	9	0.9630	0.07349	0.000
	GCL1	17	0.6275	0.23221	0.085
	Lego	14	0.8571	0.22746	0.000
Confidence	Scratch	8	0.9688	0.05786	0.000
	GCL2	9	1.0000	0.00000	-
	GCL1	17	0.5000	0.30619	0.234
	Lego	14	0.8571	0.25409	0.000
Learnability	Scratch	8	0.9063	0.12939	0.000
	GCL2	9	0.8889	0.22048	0.000
	GCL1	17	0.4412	0.42875	0.023
	Lego	14	0.7500	0.37978	0.000
Likeability	Scratch	8	1.0000	0.00000	-
	GCL2	9	1.0000	0.00000	-
	GCL1	17	0.9412	0.24254	0.000
	Lego	14	0.9643	0.13363	0.000

Table 4: Test of Homogeneity of Variances

Characteristic	Levene Statistic	df1	df2	Sig.
Age	0.353	3	44	0.787
Profile	1.758	3	44	0.169
Experience	3.588	3	44	**0.021**
Tendency	0.537	3	44	0.660
Effectiveness	6.975	3	44	**0.001**
Satisfaction	2.892	3	44	**0.046**
Confidence	8.079	3	44	**0.000**
Learnability	2.828	3	44	**0.049**
Likeability	1.711	3	44	0.178

5.2 Dataset preparation

The video recordings were analysed following a protocol previously established by the research team. Apart from (i) assessing the success, we checked if the team (ii) reused the concepts from previous exercises, (iii) experienced technical problems or functional errors, (iv) had interaction difficulties (e.g. using copy/paste for visual objects or connecting the same objects in sequence), (v) reused previously constructed sequences (within the same exercise), or (vi) used any other additional language features (e.g. zooming). The remaining data was extracted from the questionnaires [4].

5.3 Hypotheses testing

For testing our hypotheses, we used the Welch t test, as it is robust to deviations from the normal distribution, different sample sizes and different variance in the samples, thus following the recent recommendations on data analysis for Software Engineering empirical evaluations (which summarises best practices in statistical analysis on other domains) [17]. Table 5 summarizes the results of these tests. Note that, because *Confidence* and Likeability had a constant value (the top possible score for each of them, the corresponding lines are not filled in table 5.

Table 5: Welch t test scores

	Statistic	df1	df2	Sig.
Age	.170	3	21.109	.915
Profile	.423	3	20.891	.739
Experience	.921	3	21.381	.448
Tendency	.527	3	20.597	.669
Satisfaction	10.723	3	24.108	**.000**
Confidence	-	-	-	-
Learnability	5.678	3	23.334	**.005**
Likeability	-	-	-	-
Effectiveness	10.886	3	20.483	**.000**

We used a Games-Howell post-hoc test to determine which languages were significantly different from which languages, according to our set of characteristics under scrutiny. Table 6 summarises this test's result, for the comparisons involving either GCL1 or GCL2, or both. We observe that GCL1 lead to a significantly lower *Satisfaction* and *Effectiveness* when compared to Scratch, GCL2 and Lego. GCL1 was also significantly harder to learn than Scratch and GCL2, but not significantly different when compared to Lego. In contrast, GCL2 was consistently as good as Lego and Scratch, while superior to GCL1 in terms of Satisfaction, Learnability and Effectiveness.

Table 6: Games-Howell test

Characteristic	(I)Tool	(J)Tool	MD(I-J)	Std. Err.	Sig.
Satisfaction	GCL2	Lego	.10582	.06554	.397
	GCL2	Scratch	.00463	.03216	.999
	GCL2	GCL1	.33551	.06142	**.000**
	GCL1	Lego	-.22969	.08287	**.046**
	GCL1	Scratch	-.33088	.06005	**.000**
Learnability	GCL2	Lego	.13889	.12531	.689
	GCL2	Scratch	-.01736	.08657	.997
	GCL2	GCL1	.44771	.12734	**.009**
	GCL1	Lego	-.30882	.14531	.169
	GCL1	Scratch	-.46507	.11360	**.003**
Effectiveness	GCL2	Lego	.16984	.09242	.293
	GCL2	Scratch	.15556	.08948	.362
	GCL2	GCL1	.57908	.10264	**.000**
	GCL1	Scratch	-.42353	.12967	**.018**
	GCL1	Lego	-.40924	.13171	**.021**

*RQ1: How does the current version of GCL (GCL2) compare to the used baselines (GCL1, Lego and Scratch) regarding the **Effectiveness** of the children when programming a robot?*

As seen in table 5, there was a statistically significant difference among languages, with respect to the overall *Effectiveness*, with GCL2 (M=.9556; SD=.08819), Scratch (M=.8; SD=.23905) and Lego

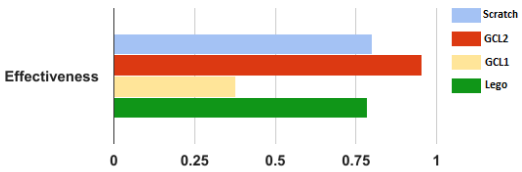


Figure 11: Effectiveness results

(M=.7857; SD=.32783) clearly allowing the teenagers using it to outperform those using GCL1 (M=.3765; SD=.40548), as detailed in table 6. This suggests that **GCL2 has achieved a similar level of *Efficiency* compared to the two commercial baseline languages, and has significantly improved when compared to its previous iteration, GCL1, in this aspect**. Figure 11 illustrates this. As such, we can reject our null hypothesis $H_0 1$.

*RQ2: How does the current version of GCL (GCL2) compare to the used baselines (GCL1, Lego and Scratch) regarding the **Satisfaction** of the teenagers when programming a robot?*

As seen in table 5, there was a statistically significant difference among languages, on the overall satisfaction, with Scratch (M=.9583; SD=.05893), GCL2 (M=.963; SD=.07349) and Lego (M=.85710; SD=.22746) providing a higher satisfaction than GCL1 (M=.62750; SD=.23221). This suggests that **GCL2 has achieved a similar level of *Satisfaction* compared to the two commercial baseline languages, and has significantly improved when compared to its previous iteration, GCL1, in this aspect**. Figure 12 illustrates this. As such, we can reject our null hypothesis $H_0 2$.

We can further break down this observation with a closer look to the components of *Satisfaction*: *Confidence*, *Learnability* and *Likeability*. *Confidence* obtained a perfect score (M=1.0; SD=.0) for GCL2, closely followed by Scratch (M=.96880; SD=.05786) and Lego (M=.8571; SD=.25409), but contrasting to GCL1 (M=.5; SD=.30619). Concerning *Learnability*, Scratch (M=.90630; SD=.12939) and GCL2 (M=.8889; SD=.22048) had a statistically significantly higher score than GCL1 (M=.4412; SD=.42875), but not significantly higher than Lego (M=.75; SD=.37978). Finally, the perfect scores of Scratch and GCL2 conderning *Likeability* (M=1.0; SD=.0) were not statistically significantly different from those of Lego (M=.9643; SD=.13363) and GCL1 (M=.9412; SD=.24254).

Participants Background:

We also need to stress that the observed differences are **not** attributable to different participant backgrounds. Indeed, the results from the background questionnaires indicate a comparable profile of participants for all languages. This is visible from table 5, where none of the properties *Age*, *Profile*, *Experience* and *Tendency* are statistically significantly different for any of the languages, i.e., they are essentially comparable.

Most participants had some experience in playing computer games, but very few of them had previously programmed a robot (see Figure 13). Some of the participants also had some knowledge of programming. These three factors gave an average Experience score for teams. The motivation to participate in the challenge was high. Most subjects expressed their tendency to learn to program,

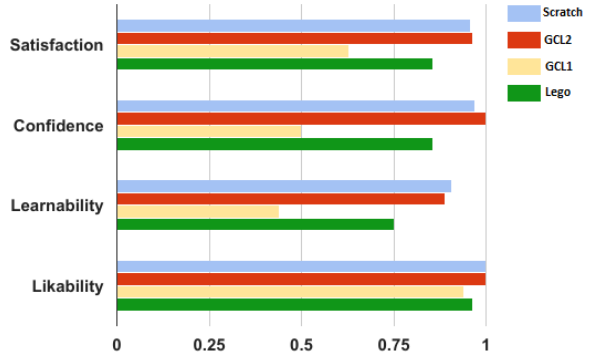


Figure 12: Satisfaction feedback

mathematics and physics, leading to a high Tendency score for all groups. The Profile score, calculated as an average of Experience and Tendency, indicates balanced teams regarding their background.

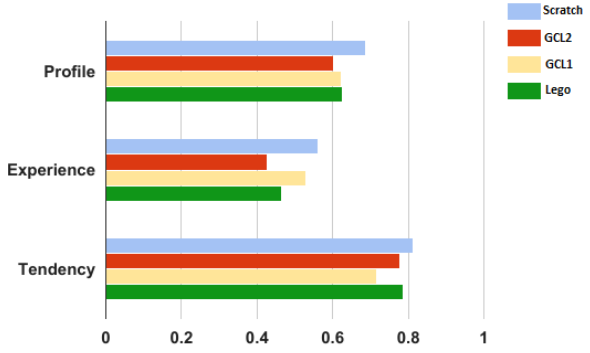


Figure 13: Profile analysis

6 DISCUSSION

We presented a systematic usability evaluation of the *GCL* language. In this section, we discuss the contributions to the software language engineering process, evaluation results and finally the implications for the further development.

6.1 Contributions to the development cycle

By designing the systematic experimental study, it was necessary to describe the context of use of *GCL* during the experiment and its goals explicitly. We have identified "Who will use the language?" and characterised the intended group of end users by the Profile, where each of the relevant characteristics was measured with a particular set of questions and can be further reused and analysed for selection of experimental subjects and specification of *GCL*'s audience. We explicitly tackle the question "Where will the language be used?" by defining its technical, social and physical environment. Also, we systematically analysed the working environment of the *GCL*'s competitors' tools to further identified both limitations and

advantages of *GCL*. Further, we explicitly expressed "How will the language be used?" through user stories. (see [4]). Developers performed the functional testing of these user stories, and the teaching artefacts (i.e. presentation documents and videos) were produced, which can be now be reused as a part of language documentation. The readability and understandability of these presentation materials were validated during pilot experiment sessions and materials can be reused in further evaluations.

Additionally, we address the question "Why will the language be used?" by defining the usability requirements and their GQM definition. Our high-level goal to evaluate usability is context-dependent, and therefore we see that its success is linearly related to the validated requirements (e.g. effectiveness and satisfaction) and a given context (e.g. evaluating a Challenge scenario in a similar experiment with LegoMS, in the first run of the comparative evaluation, using GCL1, and Scratch in the second run of the experiment, this time competing with GCL2). The trial design was successful and can be reused and save time in further evaluation sessions. The experiment results helped in making the further design decisions and identifying the pros and cons of the previous implementation.

6.2 Evaluation results

RQ1: How does GCL2 compare to the baselines (*GCL1, Lego* and *Scratch*) in terms of the *Effectiveness* of the teenagers when programming a robot? Based on the results obtained in the first run we conclude that programming the robots with *GCL1* resulted in lower Effectiveness scores when compared to programming the robot with *Lego*. However, thanks to the evaluation with the *GCL1* prototype, we found that *GCL* was already usable to some extent by teenagers. The feedback and observations of that initial run gave us insights into particular features that should be implemented to improve *GCL*'s usability. This helped to steer the development of *GCL2*, that was then tested in the second run of the experiment, this time using *Scratch* as a competitor (although, in practice, both *GCL1* and *Lego* can be seen as competitors). The feedback collected in the previous iteration proved extremely valuable, leading to a significant improvement of the *effectiveness* from *GCL1* to *GCL2*, which is now on par to *Lego* and *Scratch*.

RQ2: How does *GCL2* compare to baselines (*GCL1, Lego* and *Scratch*) in terms of the *Satisfaction* of the teenagers when programming a robot? Results showed that both *GCL1* and *Lego* were rated as satisfactory, with *Lego* providing a more intense satisfaction level. This shows that, although not being as successful as with *Lego*, the teams solving the challenge using *GCL1* still had an entertaining and motivating experience. Concerning *Satisfaction*, *GCL2* is significantly more competitive than *GCL1* and on par to *Scratch* and *Lego*.

6.3 Implications for the development of *GCL*

As a consequence of the analysis of all the data collected and the observation of the interaction of the experiment's participants with the *GCL* language, the *GCL* development team identified that new features are needed, and they will be included in future releases.

The focus is on improving the following areas:

Error Prevention and program readability needs: 1) Improve errors' feedback (including suggestions on how to solve the problems found); 2) When a user selects a new node to add to the diagram, the nodes that are possible to connect to the new node should become highlighted; 3) The users should be able to create blocks of tree concepts (this will promote reuse and facilitate the sharing and introduction of complex behaviours to novice users); 4) To prevent misinterpretations of the execution of a sequence node, an auto-alignment of new nodes added to a sequence node feature has to be introduced; and, 5) Icon Labels should be presented on mouse-over events (this should improve the user recognition of the available options).

Diagram navigation: A *fit to screen* of the entire program or only the selected nodes should be available; an improved tree collapse and expand feature (the former approach, using keys "+" and "-" was not clear to the user) should be added.

7 THREATS TO VALIDITY

Usually, it is not safe to rely on teenagers self-rating questions. To maximise the reliability of the responses, we had adult helpers interviewing them. This helped to ensure the validity and integrity of results and gave strength to design recommendations or decisions.

We did not have participants using both languages. Although this design prevents learning effects, it does create the risk of, by accident, having more "competent" teams using one language, or the other. However, by obtaining similar background scores (Profile and Age) in the four groups, we are confident that this threat was mitigated.

We compared two approaches with different robots, in the first experiment run: one using *Arduino*, and the other using *Lego* hardware. This might have introduced a bias in the results if one of the robots was easier to program than the other for some reason not directly associated with the programming language. To mitigate this threat, the second run of the evaluation contrasted *GCL2* with the Scratch for *Arduino* language. So, in this second run, the robot was the same for both languages (and also the same used with GCL1).

The choice of *LegoMS NXT* as the platform to test against *GCL1* can also be regarded as a potential threat. Lego recently released the *Mindstorms EV3* platform that introduces improvements in their development software and robots. However, in the second run of the experiment, we used a different, but also quite popular language (although not EV3), so *GCL2* is progressively being compared to other alternative languages, rather than just to *LegoMS NXT*. This diversity is expected to mitigate the effects of using a single comparison point.

8 CONCLUSION

We reported how we involved teenagers, the end users, in several iterations of the engineering process of a programming language for low-cost robots and performed usability studies. This language, *GCL*, was contrasted with *LegoMS* and *Scratch*.

The evaluation described in this paper, thanks to the involvement of the end-users (teenagers) since early stages of the development process of the language, was helpful to timely detect, prioritise, and improve crucial usability aspects of *GCL* by identifying its strengths and weaknesses. We observed the convergence of the visual language to the degree of usability (regarding satisfaction

and effectiveness) achieved by existing mature and commercial languages.

As future work, we intend to apply the same described technique in the development of DSLs for other purposes and for different end-user profiles. Also, we foresee the need for developing tools to support software language developers to deal with the significant overhead of the assessments and track the improvements on the different features of the language.

ACKNOWLEDGMENTS

The authors would like to thank the NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MA Project (Grant: FCT/MCTES TUBITAK/0008/2014).

REFERENCES

[1] Marco Angelini, Nicola Ferro, Giuseppe Santucci, and Gianmaria Silvello. 2014. VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis. *Journal of Visual Languages & Computing* 25, 4 (2014), 394–413.

[2] Arduino. 2017. Arduino. (2017). Retrieved November 30, 2017 from http://www.arduino.cc/

[3] Ankica Barišić, Pedro , Vasco Amaral, Miguel Goulão, and Miguel Pessoa Monteiro. 2012. Patterns for Evaluating Usability of Domain-Specific Languages. In *Proc. 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012*. ACM, Tucson, Arizona, 14:1–14:34. http://dl.acm.org/citation.cfm?id=2821679.2831284

[4] Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2017. Gyro Creator Language - Companion Site. (2017). Retrieved November 30, 2017 from https://sites.google.com/view/vl-empiricalstudy/home

[5] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 2001. Goal Question Metric Paradigm. *Encyclopedia of Software Engineering* 1 (2001), 528–532.

[6] Emanuela Bauleo, Serena Carnevale, Tiziana Catarci, Stephen Kimani, Mariano Leva, and Massimo Mecella. 2014. Design, realization and user evaluation of the SmartVortex Visual Query System for accessing data streams in industrial engineering applications. *Journal of Visual Languages & Computing* 25, 5 (2014), 577–601.

[7] Natacha Borgers, Edith De Leeuw, and Joop Hox. 2000. Children as respondents in survey research: Cognitive development and response quality. *Bulletin de methodologie Sociologique* 66, 1 (2000), 60–75.

[8] Amy Bruckman and Alisa Bandlow. 2002. Human-Computer Interaction for Kids. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, Julie Jacko and Andrew Sears (Eds.). Lawrence Erlbaum and Associates, Boca Raton, FL, USA.

[9] Lejos Community. 2017. Lejos. (2017). Retrieved November 30, 2017 from http://www.lejos.org/

[10] Wanda P Dann, Stephen Cooper, and Randy Pausch. 2011. *Learning to Program with Alice (w/CD ROM)*. Prentice Hall Press, Upper Saddle River, NJ, USA.

[11] Google. 2017. Blockly. (2017). Retrieved November 30, 2017 from https://developers.google.com/blockly/

[12] Maria Hatzigianni and Kay Margetts. 2012. 'I am very good at computers': young children's computer use and their computer self-esteem. *European Early Childhood Education Research Journal* 20, 1 (2012), 3–20.

[13] International Standard Organization. 2011. ISO/IEC FDIS 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. (March 2011). Retrieved November 30, 2017 from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[14] Jared Jackson. 2007. Microsoft robotics studio: A technical introduction. *Robotics & Automation Magazine, IEEE* 14, 4 (2007), 82–87.

[15] J. Johnson. 2003. Children, robotics, and education. *Artificial Life and Robotics* 7, 1 (2003), 16–21.

[16] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 83–137.

[17] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust statistical methods for empirical software engineering. *Empirical Software Engineering* 22, 2 (2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[18] Frank Klassner and Benjamin Schafer. 2014. Using the New Lego MindStorms EV3 Robotics Platform in CS Courses (Abstract Only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 745–746. https://doi.org/10.1145/2538862.2539024

[19] Roxane Koitz and Wolfgang Slany. 2014. Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU '14)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/2688204.2688209

[20] Tomaž Kosar, Marjan Mernik, and Jeffrey Carver. 2012. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering* 17, 3 (2012), 276–304. https://doi.org/10.1109/MS.2003.1231149

[21] Lego. 2017. Enchanting. (2017). Retrieved November 30, 2017 from http://enchanting.robotclub.ab.ca/

[22] Lego. 2017. Lego Mindstorms. (2017). Retrieved November 30, 2017 from http://mindstorms.lego.com/en-us/Software/Default.aspx

[23] Pedro Leonardo. 2013. *Child Programming : An adequate Domain Specific Language for programming specific robots*. Master's thesis. Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

[24] Xiao-Wen Terry Liu. 2005. *An intuitive and flexible architecture for intelligent mobile robots*. Ph.D. Dissertation. The University of Manitoba.

[25] Panos Markopoulos and Mathilde Bekker. 2003. On the assessment of usability testing methods for children. *Interacting with computers* 15, 2 (2003), 227–243.

[26] Myles McNally, Michael Goldweber, Barry Fagin, and Frank Klassner. 2006. Do Lego Mindstorms Robots Have a Future in CS Education?. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '06)*. ACM, New York, NY, USA, 61–62. https://doi.org/10.1145/1121341.1121362

[27] NCH Software. 2016. Debut Video Capture Software. (2016). Retrieved November 30, 2017 from http://www.nchsoftware.com/capture/

[28] Donald A Norman and Stephen W Draper. 1986. *User centered system design*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA.

[29] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., United Kingdom.

[30] Pololu. 2017. Pololu 3pi. (2017). Retrieved November 30, 2017 from https://www.pololu.com/product/975/

[31] Uvais A Qidwai. 2007. A LAMP-LEGO Experience of Motivating Minority Students to Study Engineering. *SIGCSE Bull.* 39, 4 (Dec. 2007), 41–44. https://doi.org/10.1145/1345375.1345411

[32] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[33] Eric Rosenbaum, Evelyn Eastmond, and David Mellis. 2010. Empowering Programmability for Tangibles. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '10)*. ACM, New York, NY, USA, 357–360. https://doi.org/10.1145/1709886.1709974

[34] Gavin Sim, Brendan Cassidy, and Janet C. Read. 2013. Understanding the Fidelity Effect when Evaluating Games with Children. In *Proc. 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 193–200.

[35] Gavin Sim and Matthew Horton. 2012. Investigating Children's Opinions of Games: Fun Toolkit vs. This or That. In *Proceedings of the 11th International Conference on Interaction Design and Children (IDC '12)*. ACM, New York, NY, USA, 70–77. https://doi.org/10.1145/2307096.2307105

[36] Karel Vredenburg, Ji-Ye Mao, Paul W. Smith, and Tom Carey. 2002. A Survey of User-centered Design Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*. ACM, New York, NY, USA, 471–478. https://doi.org/10.1145/503376.503460

[37] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (March 2006), 33–35. https://doi.org/10.1145/1118178.1118215

ANNEX V

Case study: DSE Merge

In this Annex we include the article [15], named **'STSM Report: Evaluating the efficiency in use of search-based automated model merge technique'**, which was published in COST Action IC1404 MPM4CPS in 2016.

We evaluated the tool support developed within this project at Budapest University of Technology and Economics, named DSE Merge (Section 9.5), which presents a novel search-based automated model merge [115]. Main contribution was illustration of our experiment design proposed in Section 4.1 as a part of our research process (Figure 1.1). We showed that design can be easily repeated and reused. Final report of this project can be downloaded from public repository [1].

This work was supported by COST Action IC1404 MPM4CPS.

[1] goo.gl/Kq3R1G (accessed September 19, 2017)

STSM Report: Evaluating the efficiency in use of search-based automated model merge technique

Ankica Barišić
NOVA Laboratory for Computer Science and Informatics
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, Portugal
Email: a.barisic@campus.fct.unl.pt

I. Purpose of the Visit

This report contributes to WG2, by evaluating the adequacy of the technology for model differencing and merging, which resulted from the challenge of the increasing demand for collaboration features in industrial applications of model-driven engineering (MDE).

The FP7 project MONDO, developed at Budapest University of Technology and Economics, aims to tackle the challenge of scalability in MDE in a comprehensive manner by developing the theoretical foundations and an open-source implementation of a platform for scalable modeling and model management. This technique uses rule-based design space exploration to search the space of solution candidates that represent conflict-free merged models. The approach allows engineers to easily incorporate domain-specific knowledge into the merge process to provide better solutions.

We systematically evaluate the efficiency of the technique from the user point of view using a reactive experimental software engineering approach. In particular, we asked users to merge the different versions of same model. These empirical tests include the involvement of the intended end users (i.e. engineers), which are expected to confirm the impact of design decisions. The experiment participants were observed while performing the tasks of different complexity. Evaluation took place at a the Budapest University of Technology and Economics.

Achieving scalability in modelling and MDE involves being able to construct large models and domain-specific languages in a systematic manner, enabling teams of modellers to construct and refine large models in a collaborative manner, advancing the state-of-the-art in model querying and transformations tools so that they can cope with large models (of the scale of millions of model elements), and providing an infrastructure for efficient storage, indexing and retrieval of large models.

To address these challenges, MONDO brings together partners with a long track record in performing internationally-leading research on software modelling and MDE, and delivering research results in the form of robust, widely-used and sustainable open-source software, with industrial partners active in the fields of reverse engineering and systems integration, and a global industry consortium including more than 400 organisations from all sectors of IT.

A. Technique

Industrial applications of MDE to develop large and complex systems resulted in an increasing demand for collaboration features. However, use cases such as model differencing and merging have turned out to be a difficult challenge, due to

- the graph-like nature of models, and
- the complexity of certain operations (e.g. hierarchy refactoring) that are common today.

MONDO European FP7 project [14] aims to tackle the challenge of scalability in MDE in a comprehensive manner by developing the theoretical foundations and an open-source implementation of a platform for scalable modelling and model management.

The tool support developed within this project at Budapest University of Technology and Economics, named DSE Merge, presents a novel search-based automated model merge [11] which builds on off-the-shelf tools for the model comparison step, but uses guided rule-based design space exploration (DSE) [10] for merging models. In general, rule-based DSE aims to search and identify various design candidates to full certain structural and numeric constraints. The exploration starts from an initial model and systematically traverses paths by applying operators. In this context, the results of model comparison will be the initial model, while a target design candidates will represent the conflict-free merged model.

While many existing model merge approaches detect conflicts statically in a preprocessing phase, this DSE technique carries out conflict detection dynamically, during exploration time as conflicting rule activations and constraint violations. Then multiple consistent resolutions of conflicts are presented to the domain experts. This technique allows to incorporate domain-specific knowledge into the merge process by additional constraints, goals and operations to provide better solutions.

B. Evaluation approach

Practitioners are still experiencing problems in order to adopt modeling techniques, in practice. Among other factors,

developers seem to underestimate the importance of really aligning the domain-specific support with the needs of their end users. We argue that for this kind of techniques the measure of success has to be captured by assessing the impact of using the technique, in a realistic context of use, by its target domain users. Investment into this assessment, commonly called Usability evaluation, is justified by reduction of development costs and increased revenues for other software products, brought by an improved effectiveness and efficiency by their end users [13].

Existing Experimental Software Engineering techniques [9] combined with Usability Engineering techniques [15] can be adopted in order to support this evaluations. This includes application of reactive experimental approaches, based on which the support should be tested empirically with humans using systematic techniques to confirm the impact of design decisions on usability of approach.

The proposed evaluation approach is illustrated by a real life case study of the usability evaluation of a domain-specific language (DSL) for the High Energy Physics [8]. It is also applied in the context of iterative development of a DSL for humanitarian campaigns flow specification (FlowSL) [6]. Finally, the approach is being applied in a context of DSL summer schools and in several master theses developed at NOVA University at Lisbon, involving industrial partners, among which we can highlight the example of developing and evaluating DSL that is meant to enable the children to program the robots [12].

II. Work Carried out during STSM

The experiment preparation started immediately upon receiving the positive answer from STSM committee. After obtaining the information about possible availability of participants in the period from 1-15 December, planned 1-week visit was more convenient to take place during second week of December (6-13 of December). First days of visit applicant got introduced to host team and worked on validating and improving materials needed for experiment. Pilot session took place on 10th of December in the morning, while experiment itself was scheduled for 11th December in afternoon. The collected data on machines that were used during experiment was delivered by the 17th December. The development team from Budapest University rated the success of delivered projects and STSM applicant performed other result analysis during the first week of January 2016. Finally, the report was conducted and submitted by the 13 of January.

A. Experiment Preparation

The host institution provided the subjects with different level of the modelling expertise that were to participate in experiment execution. Based on participant expertise in the domain, the availability questionnaire was conducted in advance in order to profile experiment subjects and get idea about availability for experiment (see Figure 1). Meanwhile, the development team was preparing the demo for DSE Merge tool, the tasks and training material, and finally the virtual machine environment. All provided materials were verified and improved during the STSM visit. The materials were evaluated during the pilot session that took place before the experiment execution.

The participants of the pilot session were two academics that are part of the development team, although did not participate in development of the evaluated tool.

Before starting the experiment, decisions have to be made concerning the context of the experiment, the hypotheses under study, the set of independent and dependent variables that will be used to evaluate the hypotheses, the selection of subjects participating in the experiment, the experiment's design and instrumentation, and also an evaluation of the experiment's validity. Only after all these details are sorted out should the experiment be performed. The outcome of planning is the experimental evaluation design, which should encompass enough details in order to be independently replicable.

B. Experiment Objective

The goal of experiment is to answer the following research question:

- *How usable is a proposed technique for performing the model merge operations when compared to alternative?*

In particular we tested following hypothesis:

- *H1:* By using DSE Merge engineers can perform model merge operations more effectively when compared to alternative.
- *H2:* By using DSE Merge engineers can perform model merge operations more efficiently when compared to alternative.
- *H3:* By using DSE Merge engineers can perform model merge operations more satisfactory when compared to alternative.
- *H4:* By using DSE Merge engineers can perform model merge operations with less cognitive effort when compared to alternative.

C. Experiment Context

The planning of experiment started by defining explicitly the context of use for technology under evaluation, namely DSE Merge tool.

The alternative, i.e. baseline support for model merge problem that is suitable for experimental comparison is identified to be following:

- Diff Merge [3] shows all the changes to user where the changes have to be applied manually one by one. Its strength is the user-friendly UI which is very intuitive for the novice users.
- EMF Compare [2] is default comparison and merge tool in the Eclipse environment. In each steps, the tool show only a subset of the changes that the user has to apply into the merge model. Its strength is the capability of handling very complex impacts of changes.

The alternative solutions are meant to support software engineers during model merge process. The additional benefit

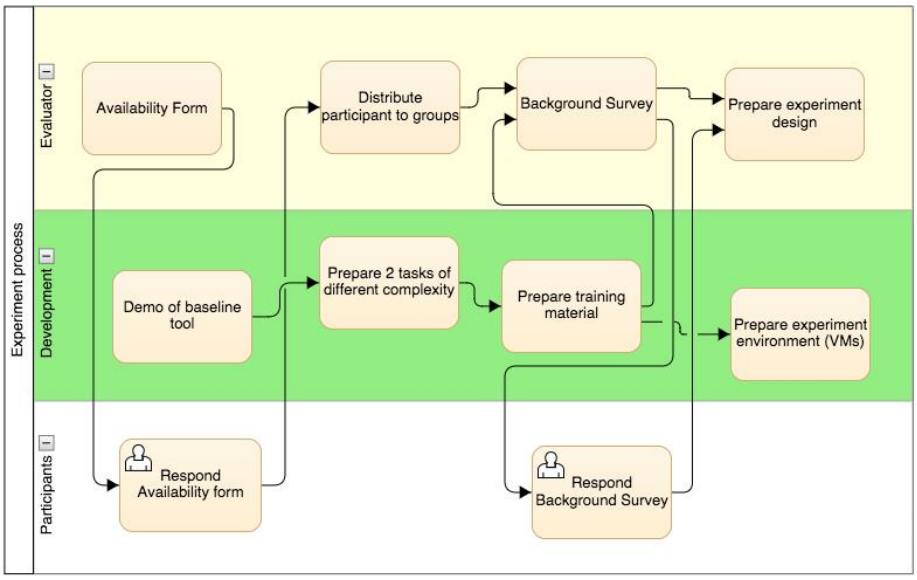


Fig. 1. Experiment preparation

claimed for the DSE Merge tool is its power to support domain experts in same process without requiring from these experts a high level of programming expertise. DSE Merge is claimed to empower incorporation of domain-specific knowledge explicitly into merge process. However, these two benefits can only be evaluated afterwards. This experiment was scoped to the similar context as alternative supports, to confirm its benefits in familiar context described as follows:

- *User Profile* - target users for this experiment are expected to be software engineers
- Technology - all three tools are ruining over Eclipse IDE. OS during evaluation was Windows 7 on Desktop computer (Intel(R) Core(TM) i5 650@3.2GHz, 8 GB RAM, 19") or Lenovo Thinkpad T61p laptop (Intel T7700@2.4GHz, 4GB RAM, 15.4").
- *Social and Physical environment* - the environment in which the tool is expected to be used reflects the typical office environment, where the user is working individually by the desk using laptop or desktop computer. Interaction is performed by use of the mouse, keyboard and the monitor.
- *Domain* - the domain meta-model that was chosen for the experiment reflected the Wind Turbine domain problem.
- *Workflow* - due to existence of the 2 different versions representing the same instance model, the user need to find the best merge solution. The problem is more complex depending on the number of the conflicts between the models. The task T0, described in the DSE-MergeWT.pdf [7] was taken as representative to problem reasoning based on domain example.

D. Training materials

Teaching session was expected to start with an *Introduction Session* to the Wind Turbine meta-model and model merge problem, that was followed with a practical reminder on basic functionality of Eclipse modeling environment. During *Introduction Session* the participants are allowed to ask questions. This session was supported by:

- Wind Turbine Control System printed document containing meta-model.
- EMF-models demo video describing use of eclipse and model merge problem.

The *Introduction Session* was followed by the *Tool Session* during which participants were not allowed to ask any question until the session is finished, for each evaluated tool. The produced materials for all three tools, DSE Merge, Diff Merge and EMF Compare were the following:

- Demo video describing the use of the tool trough presentation of the task T0 that was defined in experimental workflow context.
- Printed document containing explanations and screen shots presented in the demo video.

During the pilot session the participants were asked to give the feedback about training directly in the printed materials. The training materials were improved and can be found in folder Teaching [7]. Time was estimated to be 10 minutes for *Introduction Session*, while 5 minutes for *Tool Sessions*.

E. Experiment instruments and measurements

The experiment instruments and measurement factors are presented in Table 1.

The data for calculating the *Profile* factor was collected trough Availability and Background questionnaire. The *Profile* is influenced by following Experience factors:

- education + programming
- modelling
- EMF Compare tool

- Diff Merge tool
- DSE Merge tool
- Wind Turbine meta-model

For each Experience factor participant rated themselves by 5 point Likert scale and justify their answer by open end question. The final *Profile* score, scaling from 0-5 was calculated as average of all six Experience factors, to which it was added the value of 1 in a case that person had relevant Industry experience. In other case the person was assumed to be Academic.

The *Time* reflects the actual time taken to solve the tasks and was captured trough video analysis.

TABLE I
INSTRUMENTS AND SCALES

	Instrument	*Value*
Profile	Availability Form, Background Questionnaire	[0-5]
Time	Video recording	mm:ss
Success	Eclipse project delivery	[0-1]
Cognitive Effort	NASA TLX Scale	[0-1]
Satisfaction	Satisfaction Questionnaire	[(-1)-1]
Preference	Feedback Questionnaire	0 or 1

Success Factor is defined by following values

- 1 - if the project reflect set of correct solution and is delivered with success
- 0.5 - project delivered but is not reflecting the set of correct solutions
- 0 - no project delivery.

Quality Factor is described in following section, as it is defined specifically for each Task. The *Success* reflects the is multiplication of the Success Factor and Quality Factor.

The *Cognitive Effort* reflects the participants workload during solving task and is measured by a NASA TLX Scale [5].

The *Satisfaction* scale is reflecting average values in range (-1) strongly disagree till (1) strongly agree on a 5-point Likert scale regarding following factors:

- Easy to Use
- Confidence
- Readability and Understandability of User Interface
- Expressiveness
- Suitability for complex problems
- Learnability

The *Preference* is a factor reflecting explicit preference (marked 1) toward one of the tools used based on subset of Satisfaction criteria, that is annulled if in conflict with same factor collected using Satisfaction Questionnaire.

All defined instruments were used during the pilot session, after which trough interview the evaluator collected the suggestions and doubts regarding the surveys developed for the purpose of the experiment, and can be found in folder Instruments [7].

F. Tasks

The representative tasks, of different level of complexity (see Table II), were defined and analysed to be used during experiment execution and are documented in a Task folder [7].

TABLE II
TASK COMPLEXITY

Task	*Model Size*	*Change Size*	*Solutions*
T1	Small	4	2
T2	Small	12	8
T3	Big	6	2
T4	Big	54	¿mil

Quality Factor is defined for each task separately.

- Task 1.
  - 1 - One of the two possible solution is delivered
  - 0.75 - Only one of the two conflict resolved well.
  - 0.5 - None of the two conflicts are resolved correctly.
  - 0.25 - Other part of the model is modified.
- Task 2.
  - 1 - One of the 8 possible solution is delivered
  - 0.75 - Conflicts are resolved, but non-conflicting changes are missing.
  - 0.5 - Conflicts are not resolved, but non-conflicting changes are applied.
  - 0.25 - Other part of the model is modified.
- Task 3.
  - 1 - One of the two possible solution is delivered
  - 0.75 - Only one of the two conflict resolved well.
  - 0.5 - None of the two conflicts are resolved correctly.
  - 0.25 - Other part of the model is modified.
- Task 4.
  - 1 - At least 10 local and 10 remote changes are applied
  - 0.75 - At least 5 local and 5 remote changes are applied.
  - 0.5 - Only local or only remote changes are applied.
  - 0.25 - Other part of the model is modified.

The pilot session showed that cognitive effort is similar for each task (see Table III), probably due to impact of learning trough previous problem participants were able to solve more complex task by having similar workload. Avg time was ranging between 3-5min, while success rate was high and was a bit lower for more complex tasks.

TABLE III
TASK PILOT VALIDATION

Task	*Cognitive Effort*	*Time*	*Success*
T1	25.83	3:32	1
T2	28.61	4:59	1
T3	20.55	3:18	0.88
T4	24.02	4:27	0.83

G. Experiment Flow

The experiment took place on 11th December at the Budapest University of Technology and Economics. The general experimental process is presented in Fig.2, starting by Learning session, during which the subjects filled the Background questionnaire. After this they continue by to solve the exercises

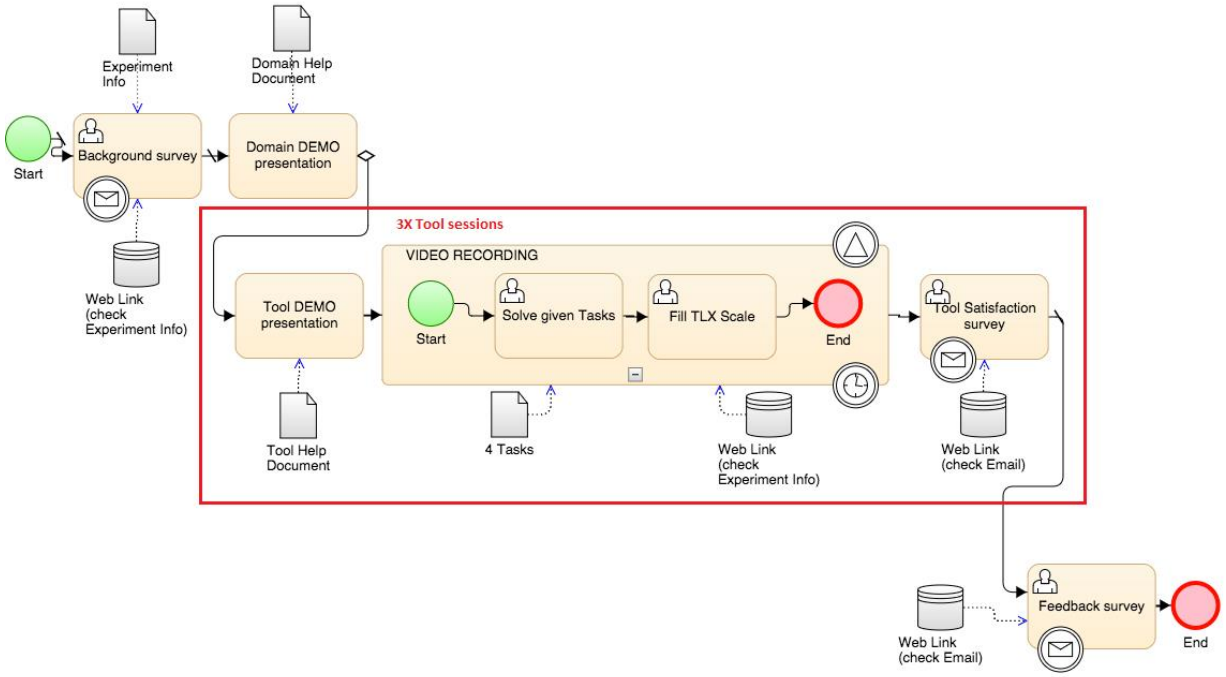


Fig. 2. Experiment treatments

during Task session, that was video recorded. Finally, during Feedback session participants filled final questionnaire rating tools that they have used. The Figure 2 except reflecting flow of activities during the experiment, explicitly shows documents and treatments that were provided to participants, as well as the instruments that were used to collect the data.

During Learning sessions the subjects learned about domain and tool. Subjects were invited to ask questions and to ask for help only after presentation. During the Task session the subjects were not limited with time to solve this tasks. They were not allowed to ask for help, except if they experienced some technical or connection problem.

During the pilot session the cognitive effort for each task was estimated to be similar, the TLX scale is decided to be used just once for each tool that is being evaluated, in the end of Video Session. Based on obtained results and opinions of the participants during Pilot Session, it was found that Diff Merge is rated as more competitive alternative for DSE Merge. The experimental groups were divided in two (G1, G2), G1 receiving first *Tool Session* for Diff Merge and then DSE Merge, while G2 opposite sequence. Finally the EMF Compare *Tool Session* was lefted to be final and was evaluated just by G1.

III. Main Results

In this section we present obtained results of the experiment.

A. Subjects background

In the Table IV we present the number of subjects and obtained Profile score and industry experience. There was total of 15 participants with. Among them around half were industry and other half academics.

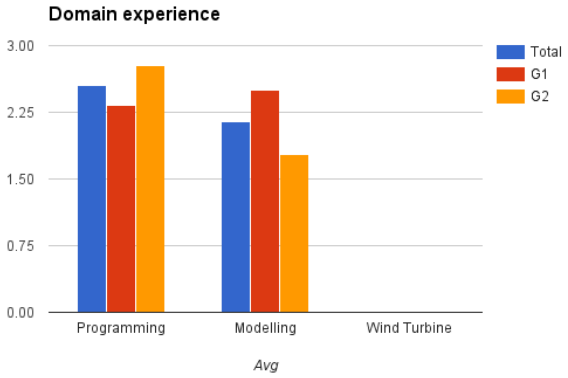


Fig. 3. Domain Experience

TABLE IV
Subject Background

	Total	*G1*	*G2*
Number of participants	15	6	9
Profile	1.65	1.92	1.39
Industry	56%	67%	44%

We can see the Experience score in Figure 3 and 4. The majority of participants had the high experience in programing and modeling. No one had experience with Wind Turbine

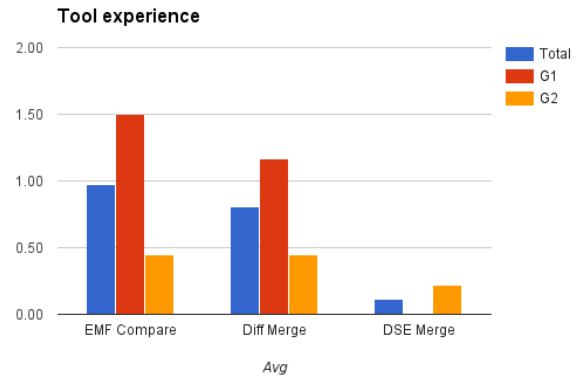


Fig. 4. Tool Experience

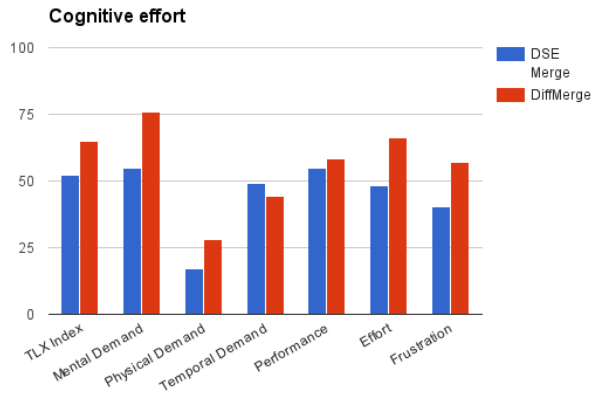


Fig. 5. Cognitive Effort

domain. Some participant had previous experience with alternative tools, while just one participant had a little knowledge about DSE Merge.

B. Comparative results

Table V presents obtained results for all three tools, for Group 1 (G1). We can see that the results confirmed that EMF Compare is candidate indicating lowest score, even as it was evaluated last, when subjects were already having high understanding of the merge process, domain and tasks and they had relevant previous experience with this tool (see Figure 4).

TABLE V
G1 RESULTS

	DSE Merge	*Diff Merge*	*EMF Compare*
Experience	0	1.17	1.5
Time	12:14	22:50	17:06
Success	0.88	0.97	0.63
Preference	5/6	1/6	0/6
TLX Index	46.65	62.83	84.93
Satisfaction	0.33	0.14	-0.36

Due to fact that Diff Merge was the object of first Task Session, while the DSE Merge of the second session, we can observe that there was a much longer time necessary to execute the tasks with Diff Merge. Success rate is higher for Diff Merge, but we can also observe that the participants did have a relative experience with this tool. On other hand they present lower cognitive effort, higher satisfaction rating and explicitly preference toward DSE Merge.

In regard to both groups, we present comparative results for DSE Merge and Diff Merge in Table VI. In total DSE Merge scored with lower time indicating a slightly better efficiency. Also, DSE Merge indicated slightly higher success rate and explicit preference by 11/15 participants, which contributes to possibility of accepting hypothesis H2. However, we could observed that there was a tendency to give the preference to the same, although it was not justify by ratings given during Tool Satisfaction Survey. This preferences were not considered. Also there were subjects that were indifferent and did not express the significant preference based on the ratings described before.

TABLE VI
DSE MERGE V.S. DIFF MERGE

	DSE Merge	*Diff Merge*
Experience	0.11	0.8
Time	20:19	23:02
Success	0.92	0.85
Preference	11	1

Concerning cognitive effort (see Figure 5), in total subjects rated with higher workload for Diff Merge regarding all factors, observing significantly higher Mental Demand and Frustration in comparison to which they experienced with DSE Merge.

We analyze more in detail the Satisfaction rating based on predefined factors in Figure 6. DSE Merge scored very high regarding easiness of use, expressiveness and learnability. Confidence was positive and better than with Diff Merge, while suitability to solve the given tasks even rated negatively for Diff Merge. User Interface, namely its readability and understandability, seems to be most important factor to be improved in order to provide better usability of the DSE MErge.

C. Threats to validity

The results presented are good indicator that DSE Merge is good enough, in regard to its purpose for people with high programming and modeling expertise. However, as it is meant to be used by the domain experts, that often are not advanced in programming, it will be necessary to evaluate it with more novice programmers, and preferably with real domain experts domain experts from a few domains, to validate the target scope of its use. Another threat was that the subjects were mostly in some way related to projects developed by the same team, which could influence their preference and satisfaction scores a bit toward DSE Merge.

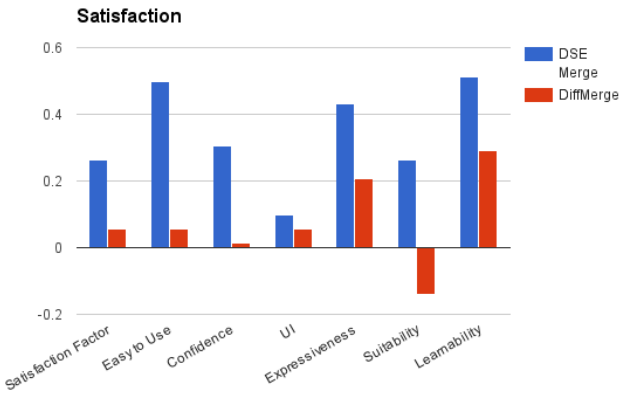


Fig. 6. Satisfaction

D. Conclusion

The most valuable contribution that resulted from this STSM visit is the experiment design, instrumentation and metrics that we believe can be easily repeated and reused for similar evaluations of new techniques for multi-paradigm modeling of cyber-physical systems. This experiment design takes deeper analysis of subject profiles, technology, social and physical environment and targeted workflow scenarios, that are defined explicitly and incorporated in a data collection instruments and reflected in hypothesis.

IV. Future Collaboration

In order to obtain significant data to confirm the experiment objective, the plan is to continue a collaboration with following goals:

1) Run provided experimental design over virtual machines, that are to be created at virtual portal of Budapest University. The undergraduate students and other possible participants will be invited to participate in experiment over portal, until we collect enough data to make a statistically relevant report. For this purpose the metrics will be standardized and calculated automatically for provided instruments. On other hand, the quality of demo videos should be improved and supported by textual explanation. Time for solving the task is to be captured trough the eclipse plug in, and experiment flow is to be preserved by use of some e-learning techniques.

2) Reusing the provided design in assessment for different tool, or the improved version of same tool with different evaluation objectives or subject profiles. This can help us to identify reusable parts, and provide scripts that can help in automatizing result analysis.

V. Foreseen publications

We plan to publish results after running the planed experiment over virtual machines in February, 2016. Target venues that we were considering are MODELS[4] and ASE[1].

References

[1] Automaed Software Engineering conference 2016, http://ase16.org/, last accessed in 11 January 2016.

[2] EMF compare, https://www.eclipse.org/emf/compare, last accessed in 11 January 2016.

[3] EMF Diff/Merge, http://eclipse.org/diffmerge/, last accessed in 11 January 2016.

[4] Model Driven Languages and Systems conference 2016, http://models2016.irisa.fr/, last accessed in 11 January 2016.

[5] NASA-TLX - Task Load Index, https://en.wikipedia.org/wiki/nasa-tlx, last accessed in 11 January 2016.

[6] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Ademar Aguiar. Introducing usability concerns early in the dsl development cycle: Flowsl experience report. In *MD^2P^2 2014–Model-Driven Development Processes and Practices Workshop Proceedings*, page 8, 2014.

[7] Ankica Barišić. Dse merge stsm experiment - https://goo.gl/Kq3R1G, last accessed in 11 January 2016.

[8] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. Quality in use of domain-specific languages: a case study. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*, PLATEAU '11, pages 65–72, 2011.

[9] Victor R. Basili. The role of controlled experiments in software engineering research. In Victor R. Basili, Dieter Rombach, Kurt Schneider, Barbara Kitchenham, Dietmar Pfahl, and Richard Selby, editors, *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, LNCS, pages 33–37. Springer Berlin / Heidelberg, 2007.

[10] Abel Hegedus, Akos Horváth, István Ráth, and Dániel Varró. A model-driven framework for guided design space exploration. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 173–182. IEEE Computer Society, 2011.

[11] Marouane Kessentini, Wafa Werda, Philip Langer, and Manuel Wimmer. Search-based model merging. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1453–1460. ACM, 2013.

[12] Pedro Leonardo. Child programming : An adequate domain specific language for programming specific robots. *MSc dissertation, Universidade Nova de Lisboa*, 2013.

[13] Aaron Marcus. The ROI of usability. In Bias and Mayhew, editors, *Cost-Justifying Usability*. North- Holland: Elsevier, 2004.

[14] MONDO. Scalable modelling and model management on the cloud, project.

[15] Jakob Nielson. *Usability Engineering*. AP Professional, 1993.

ANNEX VI

Case study: RDAL and USE-ME integration

In this Annex we include the article [25], named **'A Requirements Engineering Approach for Usability-Driven DSL Development'**, which was published in Proceedings of 10th International Conference on Software Language Engineering (SLE) in 2017.

This case study, discussed in Section 9.6, was used as a part of USE-ME implementation validation of our research process (Figure 1.1). It served as illustration of integration of USE-ME approach (Chapter 5) with RDAL requirement approach [44]. This combination of existing languages and tools provides a comprehensive requirement engineering approach for DSL development and an interesting case study of languages composition allowing the reuse of the assets of the existing languages. The approach was illustrated with the development of the Visualino (Section 9.4, Annex IV).

The project can be found at public GitHub repository[1]. Submission artefactsare published and documented as a part of conference proceeding, and installation instructions are attached after the article.

This work was supported by FCT/MEC NOVA LINCS; PEst UID/ CEC/04516/ 2013 and DSML4MA TUBITAK/0008/2014 Projects, as well as COST Action IC1404 MPM4CPS.

[1] github.com/akki55/useme (accessed September 19, 2017)

A Requirements Engineering Approach for Usability-Driven DSL Development

Ankica Barišić
NOVA-LINCS, FCT/UNL
Campus de Caparica
Caparica, Portugal 2829-516
a.barisic@campus.fct.unl.pt

Dominique Blouin
LTCI Lab, Telecom ParisTech
Université Paris-Saclay
46 rue Barrault
Paris, France 75013
dominique.blouin@
telecom-paristech.fr

Vasco Amaral
Miguel Goulão
NOVA-LINCS, FCT/UNL
Campus de Caparica
Caparica, Portugal 2829-516
(vma,mgoul)@fct.unl.pt

ABSTRACT

There is currently a lack of Requirements Engineering (RE) approaches applied to, or supporting, the development of a Domain-Specific Language (DSL) and the environment in which it is to be used. We present a model-based RE approach to support DSL development with a focus on usability concerns. RDAL is a RE fragment language that can be complemented with other languages to support RE and design. USE-ME is a model driven approach for DSLs usability evaluation which is integrable with a DSL development approach. We combine RDAL and a new DSL, named DSSL, that we created for the specification of DSL-based systems. Integrated with this combination we add USE-ME to support usability evaluation. This combination of existing languages and tools provides a comprehensive RE approach for DSL development and an interesting case study of languages composition allowing the reuse of the assets of the existing languages. We illustrate the approach with the development of the Gyro DSL for programming robots.

CCS CONCEPTS

•Software and its engineering →Software usability; Domain specific languages; Requirements analysis;

KEYWORDS

Requirements engineering, Domain-Specific language, Usability evaluation

ACM Reference format:
Ankica Barišić, Dominique Blouin, Vasco Amaral, and Miguel Goulão. 2017. A Requirements Engineering Approach for Usability-Driven DSL Development. In *Proceedings of ACM Software Language Engineering conference, Canada, October 2017 (SLE2017),* 12 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Domain-Specific Languages (DSLs) offer expressiveness for modeling systems at the proper level of abstraction, before they are automatically deployed or even simulated, with notations close to the end user domain. DSLs are designed to bridge the gap between the problem domain (essential concepts, domain knowledge, techniques, and paradigms) and the solution domain (technical space, middleware, platforms and programming languages). Bridging this gap is expected to increase productivity. However, engineers seem to underestimate the importance of aligning the languages, in particular, their notations and tools, with the skills of their end users [39]. Assessing the impact of introducing a DSL into a development process requires focusing on the productivity gains resulting from the extent to which the domain users are able to use the languages with their notations and tools [3]. Investment into this assessment, commonly called usability evaluation, is justified by the resulting reduction of development costs and increased revenues brought by improved effectiveness and efficiency of DSLs end users.

As opposed to software and systems products for which several model-based Requirements Engineering approaches have been developed showing several benefits, DSLs nowadays are, to our knowledge, mostly developed informally without such support. This situation may be due to the lack of consistent and computer-aided integration of two different and demanding complementary software processes: DSL development and usability engineering. Therefore, a more formal and iterative approach is required to develop DSLs and track all requirements, including usability requirements. As for other software products, the approach should include the context of use of the DSL in its environment, as well as the impact of recommendations with well-planned evaluation processes. Such approach can be supported by modeling all these aspects using appropriate languages and tools.

A first attempt of defining the required concepts for such approach has been made through the specification of the Usability Software Engineering Modeling Environment (USE-ME) [1, 2]. However, USE-ME focuses on usability and actually requires a complete RE process on which it can base its evaluation. The Requirements Definition and Analysis Language (RDAL) [6, 7] was developed as a fragmented language to be combined with other modeling languages in support of well known RE best practices such as those recommended by the Requirements Engineering Management Handbook (REMH) [21] and those of GORE (Goal-Oriented Requirements Engineering) [36]. RDAL was originally planned to become a standard annex of the Architecture Analysis and Design Language (AADL, SAE AS5506B standard) [1] for supporting requirements capture, analysis and verification. In the end, it led to the

SLE2017, Canada

DOI: 10.1145/nnnnnnn.nnnnnnn

[1] http://standards.sae.org/as5506b/ Accessed June 15 2017

development of the ReqSpec language and ALISA (Architecture-Led Incremental System Assurance) approach [11], which are however strongly coupled with the AADL. In contrast, the modularity of RDAL allows its reuse with other Architecture Description Languages not necessarily targeting the safety-critical embedded systems domain. Although the REMH has been written for the domain of safety-critical embedded systems, a large majority of its practices that are supported by RDAL are generic enough to be applicable to the development of many other types of systems. As a matter of fact, the concepts of the RDAL-REMH approach share many similarities with the concepts required by USE-ME.

Therefore, in this paper we present a complete framework for the RE of DSL development based on RDAL-REMH and combined with USE-ME, in order to provide a focus on the important concern of usability. This allows USE-ME to benefit from the REMH well-established practices and is also an opportunity to evaluate the planned reuse capability of RDAL for a very different domain. Furthermore, such combination of existing languages and tools constitutes an interesting case of languages composition and reuse.

In the next sections, first the RDAL-REMH and USE-ME approaches are introduced. Next, a study mapping the REMH best practices, where possible, to those of the RDAL and USE-ME approaches is presented in section 3. This study showed the need for a new DSL for the specification of DSL-based systems that we developed. This language is introduced in section 4. Next, the approach is illustrated by the modeling of the development of the Gyro DSL for programming robots.

2 BACKGROUND

2.1 RDAL-REMH

In 2008, the US Federal Aviation Administration (FAA) commanded a state-of-the-art of the RE research and a survey of the current industry practices [20]. The purpose was to select and adapt methods for the successful management, integration, verification and validation of requirements that may be developed by multiple entities. The results of that study lead to 11 best practices presented in the Requirements Engineering Management Handbook (REMH) [21]. Such practices were developed to support their incremental adoption, following a sequence tailored to each organization, in order to minimize the risk of disruption of the organization's development processes thus favoring a smoother adoption by industry of results from the RE research.

The first column of table 1 lists the 11 best practices of the REMH with a short description for each of them. Despite that the REMH was developed for the embedded systems domain, it can be observed from table 1 that a large majority of the recommended practices are not specific to this domain and can potentially be beneficial for other domains.

A combination of languages and tools was proposed to provide support of the REMH practices with models [6]. It involved the combination of the User Requirements Notation (URN, ITU recommendation Z.150 standard) [2] to model use cases, AADL for modeling safety-critical embedded systems architectures and the RDAL [7] to model and analyze requirements. This combination of languages was exercised for one of the example requirements specification provided by the REMH and illustrating its practices. The approach was very beneficial as it allowed improving significantly the quality of the specification [6]. The jUCMNav tool [3] was used for the URN language and the OSATE tool for AADL [4]. The RDAL language and its combination with URN and AADL was provided by the RDAL Tool Environment (RDALTE) [5].

While so far the fragment language RDAL has only been used with URN and AADL, it was explicitly designed to be reusable with other languages than AADL, not necessarily targeting the safety-critical embedded systems domain. Therefore, retargeting the approach for a different domain such as DSL development was of great interest in order to validate the adaptability of the RDAL-REMH approach, but also to provide model-based RE for DSL development.

2.2 USE-ME

USE-ME [1] promotes an iterative user-centered evaluation approach for DSLs. Usability evaluations are expressed as regular expert evaluator activities, in the software language engineering process. Usability engineering aims at increasing the awareness and acceptance of established usability methods among software practitioners. Knowledge of the basic usability methods is expected to enhance the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment. It is claimed to affect the user's performance and satisfaction. Further, software engineering supports the systematic development and is concerned with all aspects of software production. However, the USE-ME framework is not intended to fully support a complete development cycle. Instead, it is supposed to be integrated with existing approaches which support DSL development. For instance, a requirements engineering process for which the objective is to provide a view on usability over a complete set of requirements models. The utility specification package enables mapping to the artifacts which are commonly developed during DSL development process, like a DSL architecture, the existing goal model and requirements, or other specification diagrams (e.g. use cases, business processes).

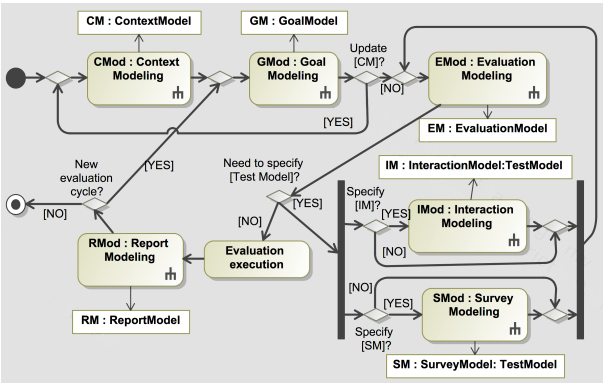


Figure 1: USE-ME activity diagram taken from [1]

[2] http://www.itu.int/rec/T-REC-Z.150/en/ Accessed June 15 2017

[3] http://jucmnav.softwareengineering.ca/jucmnav/ Accessed June 15 2017

[4] http://osate.org/ Accessed June 15 2017

[5] https://wiki.sei.cmu.edu/aadl/index.php/RDALTE/ Accessed June 15 2017

The process of usability evaluation using the USE-ME framework is presented in the activity diagram of figure 1. First, a Context Model is produced, which supports the description of the language context. The context modeling process enforces users profiling and the creation of prioritized user hierarchies, as well as modeling of the physical, technical and social environments. Further, it requires the specification and prioritization of workflow scenarios, whose actors should be already a part of a context model. The next step consists of describing the usability goals during goal modeling. While specifying the goals and their scope (e.g. by using context model instances), it is likely that new context elements are found which are relevant for the use of the DSL. In this case, it is necessary to update the Context Model and proceed with specifying a Goal Model until there is at least one usability goal for which only an actor representing an expert evaluator is responsible. This usability goal can be traced back to the functional requirements on which it depends, and its usability requirements are defined with associated metrics, which will determine success after evaluation. Further, it is necessary to define an Evaluation Model, which highlights evaluation goals and their corresponding evaluation steps. There is a *'need to specify a [Test Model]'* or reuse an existing one. The Test Model is crucial for the usability assessment process and can be defined as an Interaction Model or/and a Survey Model. These two modeling activities depend on the same Evaluation Model and should be performed in parallel in order to complement each other. However, for certain types of evaluation, it is not necessary to develop both models. For instance, when performing a heuristic evaluation, a checklist implemented as a Survey Model can be sufficient. When the Evaluation Model is ready, we can proceed with the 'Evaluation execution'. Finally, it is necessary to analyze the results of the test models and to create the Report Model, that recommends a Goal Model extension and calculates a success factor of the evaluated usability goal. Finally, it is up to all DSL stakeholders to decide to continue to *'new evaluation cycle'* or finalize the assessment period. Ideally, this decision will eventually indicate the end of the development cycle.

3 FEASIBILITY / COMPARATIVE STUDY

We first studied the concepts of both the RDAL-REMH and USE-ME approaches and compared them in order to evaluate the required effort and potential benefits of extending RDAL-REMH to support USE-ME. In some preliminary work [5] we showed that many RDAL-REMH concepts are also partially supported by USE-ME (see table 1). However in RDAL there is no specific focus on usability and usability elements can only be identified by using the RDAL user-defined category system. Nevertheless, the modeling of all other RE concerns with RDAL canl provide an essential basis for the USE-ME viewpoint on usability.

A strength of USE-ME is its usability goal coverage or achievement analysis supporting iterative development of the DSL. A similar optimization of quality attributes approach was also developed for RDAL but in the context of embedded systems development during the refinement of architecture models [22]. However, the RDAL approach does not support the notion of associated context or design of empirical studies, which are necessary for evaluation of usability. As for use cases, it was observed that USE-ME could greatly benefit from the modeling of use cases using the Use Case Maps sub-language of the URN by allowing their simulation thus complementing the corresponding Workflow concept in USE-ME.

Table 1: Mapping the REMH best practices to RDAL/AADL-UCM and USE-ME

	REMH Best Practice	RDAL-AADL-UCM Concepts	USE-ME Concepts
1	Develop the System Overview, System Context and System Goals	RDAL System Overview and Context RDAL Goals	Context Model and Usability Goal Model
2	Identify the System Boundary	AADL features from the system component identified by the RDAL system overview	N / A
3	Develop the Operational Concepts	UCM use cases	Workflows
4	Identify the Environmental Assumptions / External Entities	AADL components from the environment identified from the RDAL system overview and RDAL Assumptions	User Hierarchy and Environment Context (technical, physical and social environment)
5	Develop thje Functional Architecture and High-Level Requirements	RDAL top level of hierarchy requirements	N / A
6	Revise Architecture to meet Implementation Constraints	RDAL evolution traceability links	N / A
7	Identify the System Modes	AADL system operation modes	Scope of usability goals
8	Develop the Detailed Behaviour and Performance Requirements	RDAL requirements refining the top level requirements	Only addresses usability performance requirements
9	Develop the Software Requirements	RDAL refining specification containing requirements refining the system requirements	N / A
10	Allocate Systems Requirements to Subsystems	RDAL / AADL refining specifications	N / A, but could be considered
11	Provide Rationale	RDAL Rationale construct	Method of Usability Goal

It should also be noted that some parts of the RDAL-REMH approach are not relevant for DSL development. For example, the software requirements best practice # 9 dealing with the translation of system requirements into software requirements taking into account the different representation of system variables in software does not need to be considered. Such software requirements do not exist for DSL development.

Overall, given this comparison, we conclude that many of the features provided by RDAL-REMH can be beneficial for USE-ME as it would avoid redeveloping these constructs for the USE-ME languages. Furthermore, this comparison will be the support of a mapping defined between the concepts of the languages for their integration as described in section 5.2.

A major issue that was found, however, is the lack of an appropriate language for representing the architecture design of a DSL in a similar fashion provided by AADL for embedded systems. Reusing AADL for modeling a DSL and its environment would not be appropriate due to the difference between the domains. This triggered the development of a simple design language for the specification

of DSL-based systems, namely the DSL-based Systems Specification Language (DSSL) introduced below.

4 DSSL

The purpose of the DSSL language is to model the design of a DSL being developed and the way it is used in its environment. Goals, requirements and environmental assumptions can then be assigned to elements of this representation of the DSL under development. Furthermore, DSSL can integrate descriptions of the syntaxes of the DSL implemented as an Ecore meta-model and potentially including graphical and / or textual concrete syntax(es) respectively represented as Sirius or / and Xtext grammar models.

4.1 Declarative Specification

Because there are typically several contexts of use for a DSL, the DSSL needs to provide declaration of types for the various elements present in the contexts. Such types allow reusing the characteristics of the declared types across various contexts.

The core DSSL declarative concepts are presented in figure 2. The *DSSL specification* class is used as a root container of all elements of a DSSL specification. Such elements consist of *context specifications* and *entity types* to be instantiated in a context specification for representing the developed DSL and the entities it interacts with. Interaction capabilities with other entities are described by *interaction features* contained by entity types. An interaction feature must refer to a *reference* declared in subclasses of entity type. Such reference is used as typing for the interaction between instances of entity types in a context of use.

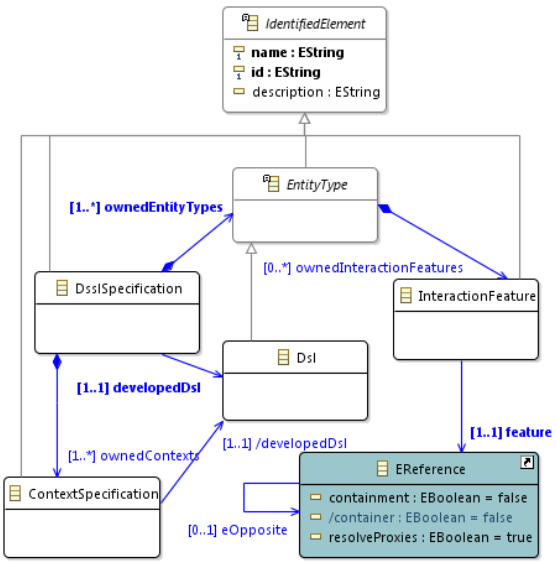


Figure 2: The core DSSL declarative elements

Subclasses of the DSSL entity type class are provided to represent *users*, *tools*, *workplaces*, *documentations* and *physical systems* (e.g. a robot) as depicted in the class diagram of figure 3. A tool can *control* or be *controlled by* other tools. Tools are further specialized into *hardware tool* and *software tool*. A software tool can support a number of DSLs and, conversely, a DSL can be supported by a number of software tools. Subclasses of software and hardware tools such as *operating systems* and *computers* are provided and declare properties specific to these elements. For instance, a computer *executes* a number of software tools, and conversely, each tool may be executed by several computers. A computer makes use of a number of *display* devices of some resolution and *color schemes*.

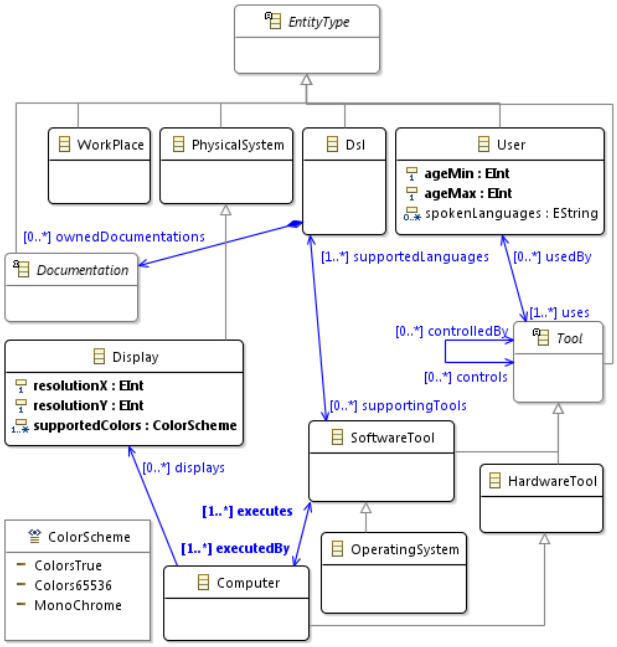


Figure 3: DSSL entity type specializations

4.2 Context Specification

The purpose of the context specification concepts is to provide means to specify the various contexts of use of the developed DSL and its interaction with entities of its environment. The DSSL context concepts are presented in figure 4. A *context specification* captures a set of *entity instances* representing the various entities involved when the DSL is used and how they interact with each other. Each entity instance refers to an entity type from the declarative specification and thus inherits its declared characteristics.

Two entity instances can be connected to each other via an *entity instance connection*, which relates a source interaction feature of the entity type of a source entity instance to a destination interaction feature of the entity type of a destination entity instance. Compatibility of the connected interaction features is checked by inspection of the type of the Ecore reference that must belong to the class of the entity type, which must be compatible with the class of the entity type of the connected entity instance (that is, they can be from either the same class or from a subclass). Examples of this are provided in section 5.1.1.

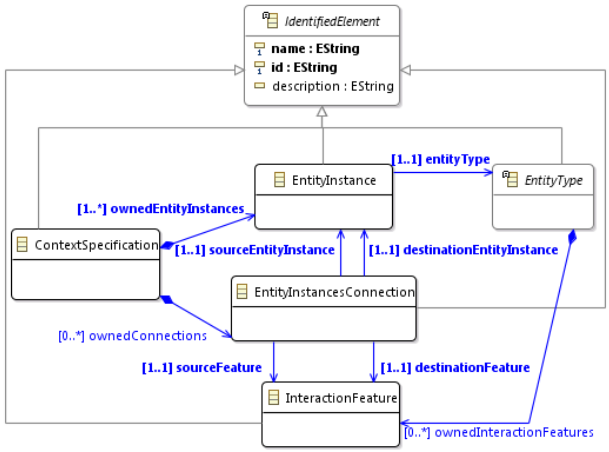


Figure 4: DSSL context instance elements

4.3 DSL Specification

The purpose of the DSL specification concepts is to allow modeling the DSL under development and its internal constituents such as its abstract and concrete syntaxes, its semantics, documentation and optionally some feature diagrams that may have served in elaborating the DSL. The DSL-related concepts are shown in figure 5.

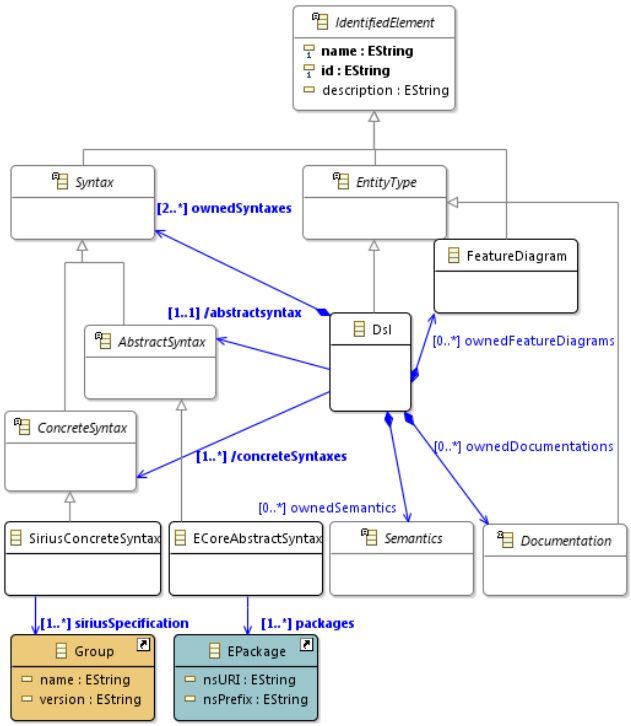


Figure 5: DSSL DSL description elements

A *Dsl* entity type owns a set of *syntaxes* providing the language vocabulary. A syntax can be of *abstract syntax* or *concrete syntax* kind according to its intended use by respectively computer programs or humans. Concrete implementations of such syntaxes must be provided to contain the DSL vocabulary that can be expressed as metamodels, grammars or graphical syntax models. The DSSL language provides default implementations classes for Ecore metamodels and Sirius diagram notation models[6].

In order to achieve this, an *Ecore abstract syntax* class can be instantiated and refer to a set of *Ecore packages* providing metamodel classes and properties for the abstract syntax of the DSL. As in this DSL we are using a graphical syntax, a *Sirius concrete syntax class* can be instantiated and refer to a set of *Sirius specifications* providing graphical notations for nodes and edges of diagrams for the DSL.

Providing such concrete syntax implementations allows for assigning requirements to the contained elements for verifying important properties such as usability. This is illustrated in section 5.1.4.

5 APPROACH

We used the Gyro DSL[7] to illustrate our approach. Gyro was selected because it was already used to validate the USE-ME framework. It is a visual language that allows children to develop programs to control Arduino robots through the manipulation of visual elements. It avoids having to program the Arduino textual code directly, which requires technical skills not owned by the vast majority of children. Gyro specifications programmed by children are automatically translated into Arduino textual code. This activity is expected to help children to quickly understand the programming mechanisms. A strong engagement factor is that children have the opportunity to observe their own developed program running on a real physical robot. This way, they receive feedback on the implications of changes in their program.

Gyro's first prototype (originally named Visualino) was developed as a joint project between Artica, a company that specialises in the development of robotic and audio-visual solutions in Lisbon and the NOVA LINCS research lab. During the development, exploratory evaluations were conducted with the objective of assessing the usability of the visual syntax provided by the language, the learnability of the associated tooling and users satisfaction, which is one of the primary characteristics to be assessed during usability evaluations.

5.1 RE for Gyro with RDAL-REMH

The requirements specification for Gyro follows the REMH best practices of Table 1. We combine RDAL, DSSL and UCM to support these practices with models. This combination of languages is achieved with dedicated traceability links starting from RDAL models to DSSL and UCM models. In order to visualize the combined models, we provide a graphical representation for the combined languages composed of the individual notations of the individual languages. Diagrams of this representation will be displayed to present the modeling the Gyro DSL development example.

[6]https://www.eclipse.org/sirius/ Accessed June 15 2017
[7]http://gyro.artica.cc/ Accessed June 15 2017

5.1.1 System Overview. The very first practice of the REMH advocates to capture a synopsis for the system to be developed including its purpose, and to provide an overview of the system and its environment for all the contexts in which it will be used. It also recommends to capture preliminary system goals.

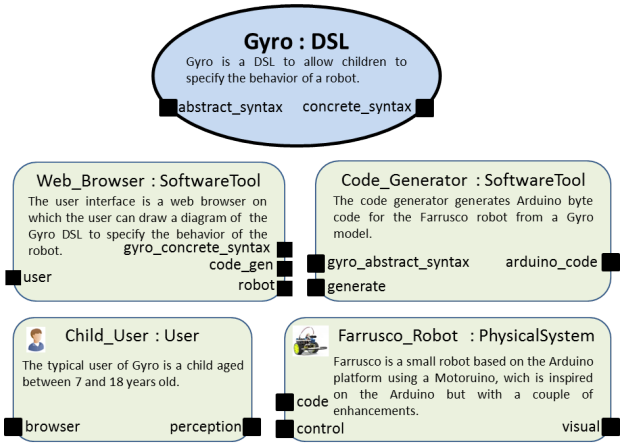


Figure 6: A system overview diagram for the the Gyro language

We consider that the system to be developed is a complete DSL including its abstract and concrete syntaxes and its semantics. The use of a DSL typically involves a user interface displaying the DSL concrete syntax and some computer program(s) that will perform some business logic with models of the DSL making use of the its abstract syntax. For the Gyro DSL, the external entities consist of a web browser serving as user interface, a child user, a code generator that transforms Gyro models into Arduino byte code and a Farrusco robot that executes the behavior programmed in Gyro. We use concepts of both RDAL and DSSL to describe these entities. On the DSSL side, instances of the *DSL*, *SoftwareTool*, *User* and *PhysicalSystem* entity types are used for that as shown in figure 6. For each entity, a description in natural language is provided and a set of *interaction features* that describe how instances of the entity types can interact with each other.

On the RDAL side, an instance of the RDAL *system overview* class is provided to identify the DSSL elements that are part of the system overview. Such system overview element is represented by the canvas on which the DSSL entity types are drawn in figure 6. The RDAL system overview is linked to the *DSSL specification* that contains the entity types. The system to be developed (the Gyro DSL entity type) is also identified via a dedicated traceability link from the RDAL *system overview* to the DSSL Gyro DSL entity type. This link also allows to compute the system boundary as the set of interaction features of the Gyro DSL entity type and stored in a property of the RDAL system overview class. The system overview class also provides additional text attributes to capture the purpose of the system and its synopsis.

System Context. The REMH recommends context diagrams for describing the different contexts in which the system is to be used in its environment. Such diagrams are captured again as a combination of RDAL and DSSL elements. On the DSSL side, a *context specification* instance is created containing a set of *context entity instances*, each of which being typed by one of the predefined *context entity types* declared in the system overview (figure 6).

A context diagram for the normal use of Gyro is shown in figure 7, where instances of the entity types of the system overview are depicted as rounded boxes connected to each other via interaction features declared in their respective types. The DSL system to be developed is highlighted by a thicker border and darker background color.

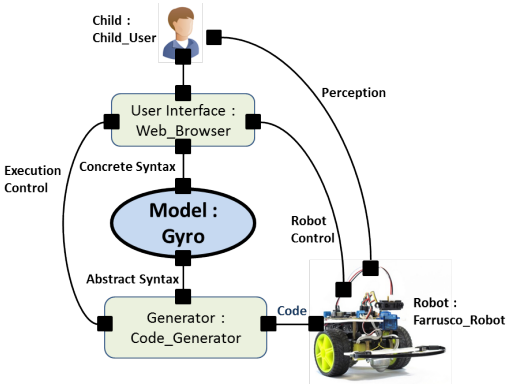


Figure 7: A context diagram for the normal use of the Gyro language

For our example, a child user interacts with the web browser serving as user interface, which itself interacts with the Gyro DSL through its concrete syntax. The Gyro DSL interacts through its abstract syntax with the computer program performing code generation for the robot. The user can trigger code generation from the user interface and then start the robot from the user interface and observe its behavior.

System Goals. System goals are captured in RDAL in terms of functional and non-functional goals. RDAL makes a clear distinction between goals and requirements. As opposed to requirements, which are either verified or not by the system in a Boolean manner, goals may be partially achievable - this is denoted with a percentage of achievement from 0% to 100%. Furthermore, goals can have unresolved conflicts, unlike requirements, since this would indicate an unfeasible system. The level of achievement and conflict among goals can be used to support design optimization and trade-offs, as in [22].

Each goal can be associated with a set of stakeholders from which it origins. A set of rationale elements can also be associated with a goal describing why the goal exists. Each rationale must however be linked to a set of stakeholders of the goal from which it originated.

Finally, RDAL goals can be linked to UCM use cases that detail how the goal is achieved with various scenarios describing the interaction of the user with the system and other entities. The advantage of such practice is to indicate why the use case exists and ensure it is a needed behavior.

For the Gyro DSL, the functional goals are:

- G1: The user should be provided with concepts to specify behaviors of robots
- G2: The teacher should be supported to create customized configurations
- G3: The software engineers should be supported to create new language components
- G4: The DSL should be easy to use for non robot specialists

RDAL goals are depicted as rounded corner boxes in the diagram of figure 8 and linked to their stakeholders and rationales, which are respectively represented as man and cloud symbols. Also represented on the diagram is the use case for programming the behavior linked to the functional goal G1 it aims to achieve. The modeling of such use cases is presented in the next section that covers the development of the operational concepts best practice of the REMH.

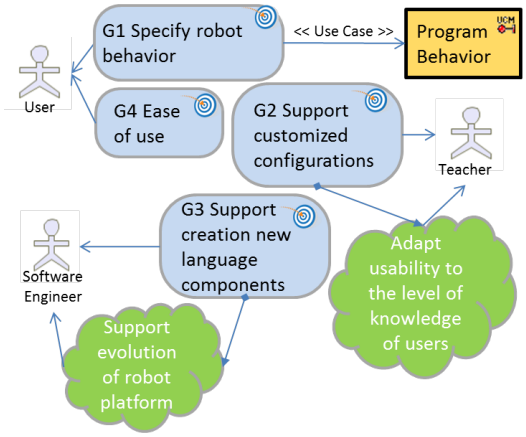


Figure 8: A diagram of the goals for the Gyro language

5.1.2 Operational Concepts. The purpose of this practice is to develop scenarios for the use of the system. The REMH suggest to capture such scenarios with use cases. We have chosen the UCM sub-language of the URN for that. UCM has the advantage of showing on the same diagram both the entities (actors or systems) and the actions that they perform in a quite compact notation (figure 9). An entity is represented by a box containing the actions (steps) performed by the entity. Actions are represented as crosses located along a path representing sequences of actions, which may have forks and joins for representing different scenarios occurring upon specific conditions. Use cases can be reused by being called from other use cases as represented by diamonds (figure 9). This is particularly useful to capture exceptions cases specifying how entities respond to exceptions in a modular way.

We present a few use case diagrams for Gyro that were developed with the jUCMNav tool. jUCMNav provides a graphical editor and a simulator for use cases scenarios.

Normal Use of the DSL. Figure 9 shows the use case for the normal use of the Gyro DSL where entities in the use case diagram correspond to entities in the context diagram of figure 7 as identified by their identical names. The use case describes a dialog between the child user and the web browser user interface. The path shows how the browser responds to actions (crosses) initiated by the child. Such responses are captured as sub-use cases describing the interaction between the browser and other entities such as the DSL model, the code generator and the robot. Pre and post conditions can be set describing the conditions under which the use case scenarios can be executed and what changes are performed after the use case.

The user first launches the Gyro programming application. The user interface then displays the start page with appropriate action menus. The user then asks to create a new robot behavior. The browser displays the programming page. The user then programs a behavior as described by a sub use case. Whe the program is completed, the user tries to execute the programmed behavior on the robot. Then, two sub cases are provided depending on whether the robot is present or not in the environment. The case when the robot is absent corresponds to an exception case deviating from the normal sunny day behavior. A specific sub use case is provided to describe how the system should handle this exception. When the behavior is correct, the user saves the model and the use case ends.

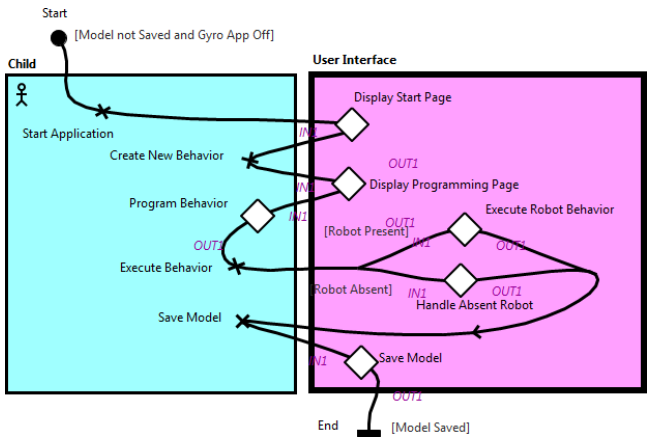


Figure 9: A UCM diagram for the normal use of the Gyro DSL

"Create Branch" Sub Use Case. The complete set of use cases for the Gyro DSL is too large to be presented in this paper. However one of the sub use cases for creating a branch in a Gyro model is presented to further illustrate the best practice. This sub use case will also be used to illustrate how requirements for the system functions can be traced to use case steps from which they were identified. The bottom part of figure 11 shows this sub use case. The user either selects a parallel or a sequential node and the user interface responds to the action by displaying the graphical concrete syntax element on the interface. The red trace on the figure represents the simulation of the scenario for the sequential node.

5.1.3 Environmental Assumptions. The REMH recommends to specify the assumptions the system makes on its environment in order to operate correctly. Identification of a system's environmental assumptions is essential for enabling the reuse of the system in different environments. It has been shown that failure to identify environmental assumptions can lead to misuse of the system and is a common cause of failure [21].

An example of a reasonable assumption for the Gyro DSL could be that the user interface can display colors. This assumption would be violated if the user interface turned out to be a basic LCD display located on the robot, for example. The Gyro DSL highly relies on such assumptions, since its concrete graphical syntax makes use of colors to indicate invalid specifications. Another assumption is that the user can speak a specific language, such as English, in which the DSL's concrete syntax is expressed.

The RDAL *assumption* construct is used for modeling environmental assumptions as shown in figure 10. The assumption is assigned to the DSSL *user* entity type of figure 6 and its constraint, expressed in the Object Constraint Language (OCL) [8] checks that English belongs to the languages spoken by the user. The rationale for this requirement is captured and linked to the software engineer stakeholder that issued this assumption based on the concrete syntax he defined.

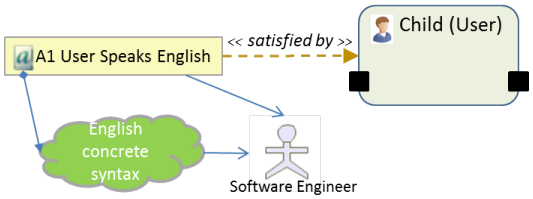


Figure 10: Example environmental assumption for the user of the Gyro language

5.1.4 System Functions. Inspired from the four variable model proposed by Parnas and Madey [28], the REMH recommends to capture a complete and consistent set of detailed system requirements that define how the system must change the variables it controls in response to changes of the variables it monitors. This shall be done for each possible system state and inputs and, if relevant, the allowed tolerance of the specified values and performance characteristics should also be specified.

Functional Requirements. Applied to DSL development, this consists of specifying what syntaxes should be defined to support the specification of the behavior of a robot. In order to do this, a hierarchy or RDAL refined requirements is defined, starting from a very high level requirement for the Gyro DSL entity type (figure 11) refined into two requirements; a requirement R1.2 stating that an abstract syntax should be provided and a requirement R1.1 stating that a concrete syntax should be provided. R1.2 is assigned to a DSSL *Ecore abstract syntax* instance that refers to an Ecore package providing the DSL classes and relationships, while R1.1 is assigned to a DSSL *Sirius concrete syntax* instance that refers to a Sirius model providing graphical syntax elements for the DSL. Both of these requirements must be verified for the refined requirement R1 to be verified.

The leaves of the requirements refinement tree (R1.2.1 and R1.2.2) are assigned to the Gyro Ecore package to check that it contains the required sequential and parallel classes. Identifying such classes is achieved by an OCL expression examining the classes of the package and checking for predefined Ecore meta annotations attached to the

[8] http://www.omg.org/spec/OCL/

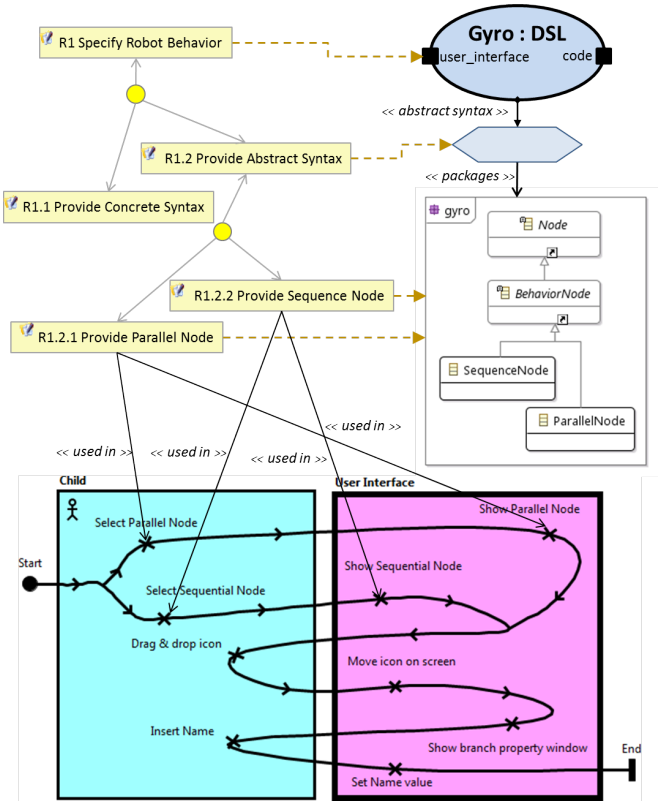


Figure 11: Example functional requirements for Gyro's abstract syntax

classes and specifying their role. Also note that R1.2.1 and R1.2.2 are also linked to steps of the UCM use cases from which their need was discovered (section 5.1.2).

Figure 12 shows similar requirements for the Gyro concrete graphical syntax specified as Sirius models. R1.1 is refined by R1.1.1 that requires that a Sirius diagram definition exists. Such requirement is refined again by a set of requirements for each required graphical element of the syntax such as nodes and edges. Those requirements are expressed by OCL queries that check for the existence of a graphical element associated with the Ecore class of the given role.

Non-Functional Requirements. The previous subsection has shown how functional requirements are captured. However non-functional requirements can be captured as well, especially those for the usability of the concrete syntax.

We illustrate this by presenting an example requirement for the consistency of a graphical notation. For instance, the graphical syntax of the AADL suffers from several inconsistencies. The virtual / abstract components are represented by a dashed border, but that same border is also used for some non-abstract components. The dashed border, which at first seems to indicate the virtuality characteristic of such components, is, after all, also used for non-abstract components, thus leading to an inconsistent and confusing notation. This problem could have been automatically detected by a

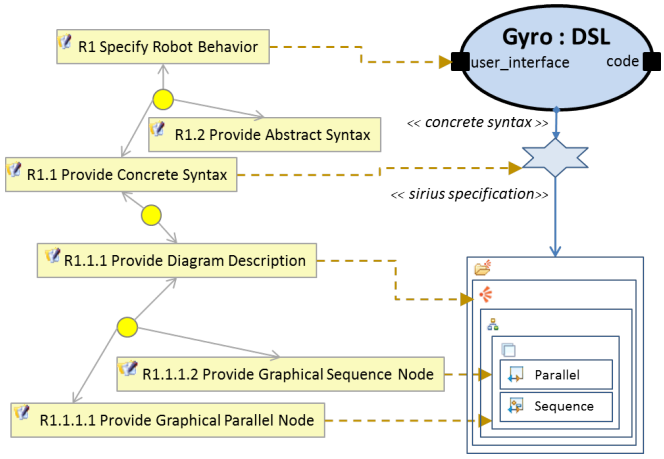


Figure 12: Example functional requirements for Gyro's concrete syntax

consistency notation RDAL requirement assigned to the graphical nodes of the notation.

Applied to Gyro, the metamodel includes a set of nodes that describe behavior such as the parallel and sequence nodes. We create a RDAL requirement that is assigned to the package containing the Gyro classes. The requirement checks for all contained classes that if the class is a behavioral node (i.e. it extends the *Behavior* class) then its corresponding Sirius notation share some visual attribute identifying the behavioral characteristic.

Many more requirements of that kind could be added, for example, to verify that best practices in defining graphical notations such as those suggested in Moody's *"Physics of Notations"* [25] are followed.

Similarly, RDAL non-functional goals and requirements are used to respectively support optimization of usability and require a minimum usability threshold for concrete syntax elements. For example, the RDAL non-functional goal G4 (figure 8) for maximizing usability is assigned to the Sirius model. The level of achievement of the goal (a number between 0% and 100%) is then valued through an annotation of the graphical syntax model following a usability evaluation with USE-ME. The result is then retrieved from the annotation by the goal's OCL expression to provide the level of achievement of the goal. Associated with this goal, we can also define a requirement that the resulting usability is greater than a given threshold to indicate if the notation is usable enough or not for the child user.

This completes the presentation of the modeling of Gyro with RDAL-REMH. In the next section, we present how USE-ME is integrated to support usability-driven DSL development.

5.2 Integrating RDAL-REMH and USE-ME for Usability-Driven Development

The RDAL-REMH approach lacks support of context-aware goal evaluation, which is necessary for usability evaluation. Usability evaluation is performed in a concrete context of use, e.g. with particular users, in a specific environment and performing selected scenarios. This means that after the validation of a usability goal and associated requirements, the results reflect only a partial scope. This is because it would be too expensive to perform evaluations taking all different combinations of the context model instances (e.g. using all possible robot configurations, testing all possible scenarios with participants, and having a significant number of participants having all possible combination of demographic and knowledge characteristics). Therefore, the USE-ME approach suggests a calculation of a *Success Coverage* of the usability goal, which will reflect the percentage of the scope which was taken in consideration during a validation, when compared to the complete context specification of the usability goal.

However, many context-related concepts and the DSL architecture are already captured in the RDAL-REMH approach described in the previous section. Therefore, it is necessary to enable usability evaluation to reuse and refer to the RDAL artifacts while performing context and goal modeling activities suggested by the USE-ME approach. On another hand, while applying the USE-ME approach it is likely that context and goal elements will be extended, or new ones discovered therefore directly contributing to the requirements refinement and DSL artifacts specification. In the following, we present an integrated process and highlight interaction points and information flow between the RDAL-REMH and USE-ME approaches (see figure 13). The idea is to connect the RDAL non-functional goals referring to usability with a 'Quality in Use' Usability Goal of the USE-ME Goal Model. This root goal represents the highest objective of the USE-ME modeling approach (usability for all possible workflows, environment combinations and profiles).

USE-ME provides a Utility Package which supports mapping of the artifacts relevant for the application of the USE-ME process. First, it is to define a DSL and refer to its abstract and concrete syntaxes, which can be done directly by correlating a DSSL Gyro specification with a *DSL* artifact of USE-ME[9] and relating a concrete/abstract syntax version in each development iteration. This helps to trace which evaluation context was considered in the USE-ME application cycle. Further, the USE-ME *DSL* has an *Existing Goal Model*, which refers to the complete RDAL requirements specification model. The functional goals are further obtained from the RDAL *Goals Package* and functional requirements from the RDAL *Requirements Package*. USE-ME expects correlation of requirements with a goal to which it contributes. Finally, the UCM use case diagrams can be mapped to the *Process Model* USE-ME artifact. After this initial mapping, we follow the USE-ME activities (see figure 1) to describe the further integration.

5.2.1 Context Modelling. During context modeling it is necessary to specify *User Profiles* and prioritize the *User Hierarchy* taking into account that there is the need to perform a usability evaluation of each of them. The USE-ME framework suggests that the first level children *User Profiles* should be referred as the DSL development stakeholders. We can obtain information about them from the RDAL *Stakeholder* definitions; namely 'Teacher', 'Software Engineer' and 'User'. Further, USE-ME suggests the addition of a new stakeholder representing a usability evaluator, as it is favorable

[9]Gyro is named Visualino in the model as specified usability evaluation was performed on a previous version of the language

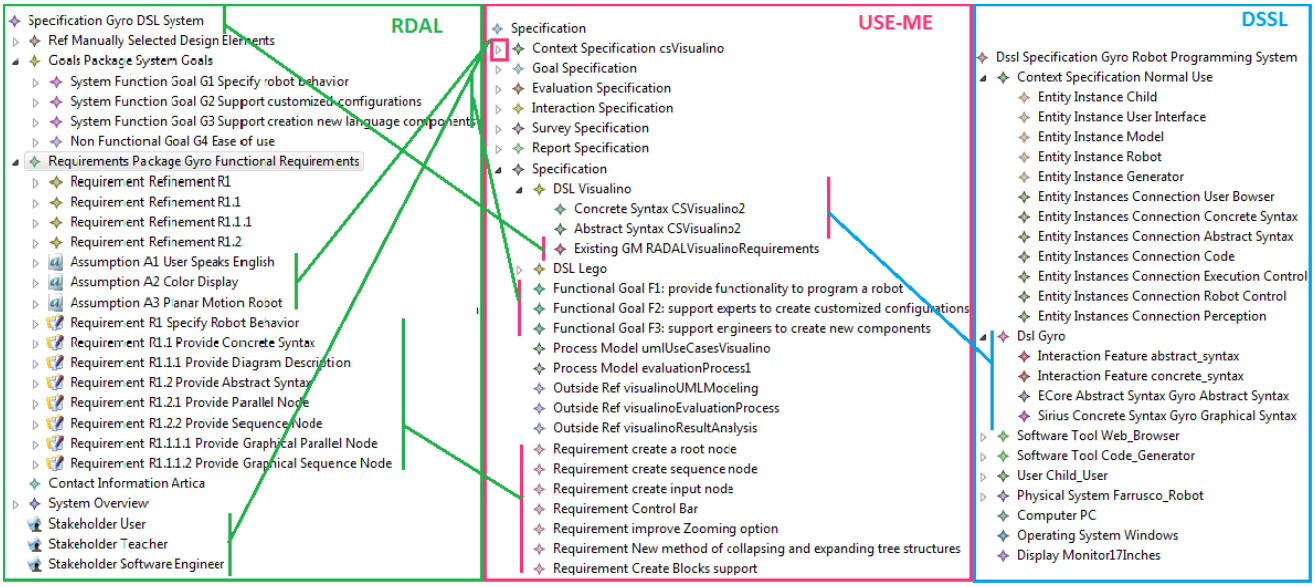


Figure 13: Integration points between RDAL, DSSL and USE-ME

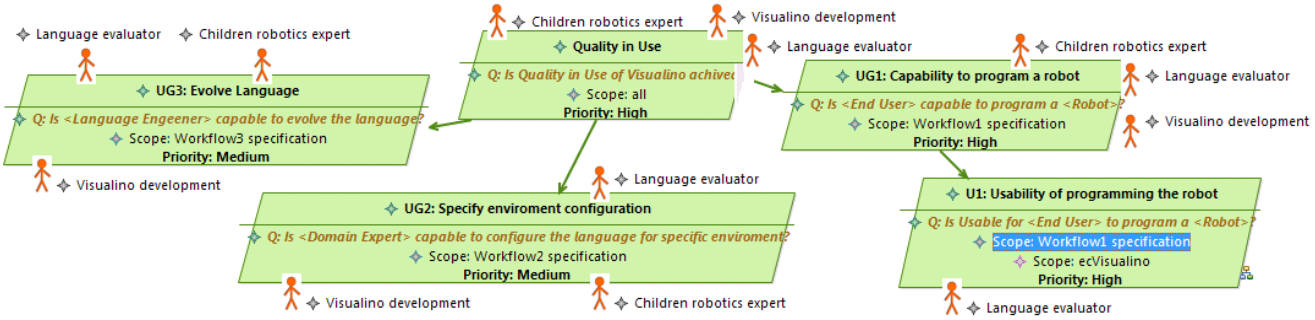


Figure 14: USE-ME Goal Model (from [1])

that evaluators do not take another role in a development project. From this initial hierarchy, the 'User' profile has a 'high' priority for usability evaluation and should be further divided into sub-profiles and characterized by expected knowledge sets and demographics constraints. We can obtain additional information from RDAL *assumptions*. For instance, the System Overview indicates that one sub-profile should be a 'Child' (figure 6) and an environmental assumption captures that the child is expected to speak English (figure 10).

The next step consists of providing the Context Environment of USE-ME. In the DSSL specification, we have context entity instances describing the considered environmental elements (figure 6). These elements, are classified in USE-ME into physical, social and technical *Environmental Variables*. For instance, a 'Web Browser' is a technical environmental variable, and it is further specified with tools which are taken into consideration (e.g. Google Chrome, Safari, etc.). The 'Robot' is a physical variable, and it details a robot version and for instance a different configuration of sensors which can be used. Finally, for instance, a social environment variable can be the 'Country' and should list the countries in which the DSL is distributed.

Finally, the context modelling ends with the creation of USE-ME workflows representing a 'high' level scenario. From the functional System Goals of figure 8, we can distinguish at least three different workflows to be considered, of which the one addressing the 'Programming a robot behaviour' should have a 'high' priority. The scenarios for the workflow can be reused from the operational concepts described by UCM use-case diagrams. USE-ME implies that actors of these scenarios (e.g. User and Web Browser) need to be specified previously as a *User Profile* or *Environmental Variable*.

5.2.2 Goal Modeling. USE-ME divides a root usability goal into sub-goals, which refer to distinct context model parts (e.g. different user profiles, environment elements or workflows), called 'scope'. As we had created three workflows from System Goals, with actors having a different 'User Profile', we had a three corresponding usability goals (figure 14). These goals are prioritized, and one having

a 'High' priority is divided into a sub-goal for which only the evaluator stakeholder is responsible. For this kind of goal it is necessary to create a *Method* which defines the measurable requirements and dependencies which impact the goal evaluation. Usually, such evaluation depends on whether the given functionality is provided, namely it relates to the *Functional Goal* from Utility, with a set of functional requirements, which will support the use case. When specifying the usability evaluation, the scenarios which can be used will depend on the functions that are provided.

One example of defining a metric for the usability goal is by use of GQM (Goal, Question, Metric) analysis [37]. For instance, one *Usability Requirement* can be a **Satisfaction**. In this case the associated question is 'How much is the {User} satisfied with {Gyro}?' and it has predefined 3 metrics: Confidence (self rated confidence score in a Likert scale), Likability (self rated likeability score in a Likert scale) and Learnability (self rated learnability score in a Likert scale). Another common indicator of usability is **Effectiveness**, which addresses the question 'Is the {User} able to correctly implement a given {Use Case}?'. It can be measured, for instance, with a percentage of correctly implemented concepts in {Use Case}.

5.2.3 Evaluation and Report Modeling. The evaluation modelling is performed only when context and goal models are completed. Evaluation specification elements are extracted mostly from existing models and instantiated into real objects. For instance, for one of the Gyro assessments where the *Usability Goal* 'U1' was evaluated, the participants were chosen to be secondary school students, as they have a 'Child' *User Profile*. The evaluation was set up to be a comparison to another DSL designed with a same purpose. To access a background of candidates, questions were directly defined from the profile characterization. The scenarios which were used for learning and testing are selected from the workflow assigned to our usability goal under evaluation. With RDAL we can easily obtain a list of the functional requirements which are satisfied, and indicate which scenarios can be performed for the current version of the language.

After evaluation execution, the Result models are obtained and USE-ME report modeling *Recommendations* are created. These recommendations contain suggestions of new requirements, which are justified by the executed evaluation model. Finally, for the evaluated goal, a success scope is calculated, which contains evaluation results and their impact depending on the size of the evaluated scope, and a real context specification associated with the goal. The recommendations can contain new suggestions for new requirements, or modification of either existing ones or of the associated context. If there are new requirements or goals to be introduced, they should be mapped back to the RDAL models.

6 RELATED WORK

Effective communication with stakeholders is extremely relevant for RE [13]. Indeed, the usability of software engineering visual languages is becoming a hot topic, building on works such as Moody's "Physics of Notations" [25]. This has inspired evaluations on the visual notations of languages such as KAOS [23], i^* [9, 27], or UML [26], and even on the way layout affects model understandability [29, 31, 32, 34]. This concern on usability is also applicable to DSLs. For example, an assessment on cognitive dimensions can be leveraged to increase DSL usability [4]. It is also common to find evaluations on DSLs using a more "traditional" empirical software engineering approach (see, e.g. [17, 19]). A more recent trend has been to directly involve the end users in the design of the DSLs [15, 16, 30, 38]. These and other usability promoting approaches can be integrated in USE-ME. The framework is open to adopting new forms of usability evaluations.

DSLs and Model-Driven Development (MDD) can be used to build frameworks for Requirements Engineering. Examples include building frameworks or derived DSLs for GORE approaches such as KAOS [12], i^* [14, 33], and model transformations between KAOS and i^* [24]. In [41] the authors build a DSL similar to Mind Maps to capture requirements and, from those, automatically derive the corresponding Feature Models[10] for variability analysis.

Kolovos *et al.* identified high-level requirements for DSLs, where usability is considered, but not regarded as a priority [18]. Indirectly, however, usability receives attention as a desirable side-effect of simplicity which, in turn, is a key quality feature for DSLs. However, evidence of using Requirements Engineering techniques and tools during the DSL lifecycle is relatively scarce, especially with usability concerns. The closest works in this direction are on the topic of Domain Engineering, also called Product Line Engineering [10], which consists of reusing domain knowledge to derive new software products. In this case, the DSLs are designed (metamodel for syntax description) and implemented, after capturing the Domain Model (core concepts), to deal with the variability and commonalities of the product's specification.

The design phase of DSLs development can be supported by capturing Domain Ontologies to derive the Language Meta-model [35]. In [40], a framework based on OWL is developed to support the design of the DSL syntax. However, these approaches do not focus on capturing the high-level goals for developing the language, and do not address the usability of the language for the end-user.

7 CONCLUSIONS AND FUTURE WORK

This paper presented a usability-driven requirements engineering approach for DSL development. The RDAL-REMH approach used for embedded systems development with the AADL language for system architectures has been adapted to DSL development by replacing the AADL with the DSSL language, a new DSL that we developed for modeling the DSL under development and its environment. We then provided a mapping between the USE-ME language and the combined RDAL-REMH languages for integrating the USE-ME usability driven development into the RDAL-REMH general RE approach. The approach has been illustrated by the development of the Gyro visual robot programming DSL. To our knowledge, no such comprehensive RE approach has been developed for DSL development, and we expect several benefits from supporting the REMH best practices with models and the integrated USE-ME usability-driven development approach.

Future work will involve completing the implementation of the graphical notation and tools for the languages and implement a view mechanism for the developed mapping between RDAL-REMH and USE-ME. The EMF Views tool [8] is a good candidate for this. Then, a comprehensive evaluation for more complex DSLs than Gyro can be performed to assess the benefits of our approach.

ACKNOWLEDGMENTS

The authors would like to thank the COST Action IC1404 Multi-Paradigm Modeling for Cyber-Physical Systems (MPM4CPS) for the context and partial support to this work, as well as NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MA Project (Grant: FCT/MCTES TUBITAK/0008/2014).

REFERENCES

[1] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2017. Usability Driven DSL development with USE-ME. *Computer Languages, Systems and Structures (ComLan)* (2017). (in press).

[2] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2017. Usability Software Engineering - Modeling Environment (USE-ME 1.1). *Faculdade de Ciências e Tecnologia, Universidade Nova da Lisboa* (2017). https://doi.org/10.5281/ZENODO.345941

[3] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. 2011. How to reach a usable dsl? Moving toward a systematic evaluation. *Electronic Communications of the EASST: 5th Int. Workshop on Multi-paradigm Modeling (MPM 2011)* 50 (2011), 13.

[4] Alan F Blackwell, Margaret M Burnett, and Simon Peyton Jones. 2004. Champagne prototyping: A Research technique for early evaluation of complex end-user programming systems. In *2004 IEEE Symposium on Visual Languages and Human Centric Computing*. IEEE, Rome, Italy, 47–54.

[5] Dominique Blouin. 2017. Report on the Short Term Scientific Mission on 'Combining Modeling Languages to Support Usability-Driven DSL Development with USE-ME'. In *Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)*. European cooperation in science and technology (COST IC1404), 1–15.

[6] Dominique Blouin and Holger Giese. 2016. Combining Requirements, Use Case Maps and AADL Models for Safety-Critical Systems Design. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 266–274.

[7] Dominique Blouin, Eric Senn, and Skander Turki. 2011. Defining an annex language to the architecture analysis and design language for requirements engineering activities support. In *2011 Model-Driven Requirements Engineering Workshop*. IEEE, 11–20.

[8] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. 2015. EMF views: A view mechanism for integrating heterogeneous models. In *International Conference on Conceptual Modeling*. Springer, Stockholm, Sweden, 317–325.

[9] Patrice Caire, Nicolas Genon, Patrick Heymans, and Daniel L Moody. 2013. Visual notation design 2.0: Towards user comprehensible requirements engineering notations. In *RE'13*. IEEE, Rio de Janeiro, Brasil, 115–124.

[10] Krzysztof Czarnecki and Eisenecker Ulrich. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA. 864 pages.

[11] Julien Delange, Peter Feiler, and Ernst Neil. 2016. Incremental life cycle assurance of safety-critical systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

[12] Patrícia Espada, Miguel Goulão, and João Araújo. 2013. A framework to evaluate complexity and completeness of KAOS goal models. In *International Conference on Advanced Information Systems Engineering*. Springer, 562–577.

[13] Daniel Méndez Fernández, S. Wagner, Marcos Kalinowski, M. Felderer, P Mafra, Vetrò, Tayana Conte, M.-T. Christiansson, C. Greer D. Lassenius, T. Männistö, M Nayabi, M. Oivo, B. Penzenstadler, Pfahl Dietmar, R. Prikladnicki, Gunther Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, José Luis de la Vara, and Roel Wieringa. 2016. Naming the Pain in Requirements Engineering - Contemporary Problems, Causes, and Effects in Practice. *Empirical Software Engineering* (August 2016), 1–36.

[14] Catarina Gralha, João Araújo, and Miguel Goulão. 2015. Metrics for measuring complexity and completeness for social goal models. *Information Systems* 53 (2015), 346–362.

[15] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2013. Enabling the collaborative definition of DSMLs. In *International Conference on Advanced Information Systems Engineering*. Springer, 272–287.

[16] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2016. Collaboro: a collaborative (meta) modeling tool. *PeerJ Computer Science* 2 (10 2016), e84.

[17] R B Kieburtz, L McKinney, Jeffrey M Bell, J Hook, A Kotov, J Lewis, D P Oliva, T Sheard, I Smith, and L Walton. 1996. A software engineering experiment in software component generation. In *Proceedings of the 18th international conference on Software engineering (ICSE'1996)*. IEEE Computer Society, 552.

[18] Dimitrios S Kolovos, Richard F Paige, Tim Kelly, and Fiona A C Polack. 2006. Requirements for domain-specific languages. In *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, Vol. 2006. 1,4.

[19] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Varanda João Maria Pereira, Matej Črepinšek, Cruz Daniela Da, and Rangel Pedro Henriques. 2010. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems* 7, 2 (2010), 247–264.

[20] David L Lempia and Steven P Miller. 2009. *Requirements engineering management findings report.* Technical Report. Technical report DOT/FAA/AR-08/34, Federal Aviation Administration.

[21] David L Lempia and Steven P Miller. 2009. Requirements engineering management handbook. *National Technical Information Service (NTIS)* 1 (2009).

[22] Grzegorz Loniewski, Etienne Borde, Dominique Blouin, and Emilio Insfran. 2013. Model-Driven Requirements Engineering for Embedded Systems Development. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 236–243.

[23] Raimundas Matulevičius and Patrick Heymans. 2007. Visually effective goal models using KAOS. In *International Conference on Conceptual Modeling*. Springer, 265–275.

[24] Rui Monteiro, João Araújo, Vasco Amaral, Miguel Goulão, and Pedro Patrício. 2012. Model-driven development for requirements engineering: The case of goal-oriented approaches. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. IEEE, 75–84.

[25] Daniel Moody. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35, 6 (2009), 756–779.

[26] Daniel Moody and Jos van Hillegersberg. 2008. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In *International Conference on Software Language Engineering*. Springer, 16–34.

[27] Daniel L Moody, Patrick Heymans, and Raimundas Matulevičius. 2010. Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. *Requirements Engineering* 15, 2 (2010), 141–175.

[28] David Lorge Parnas and Jan Madey. 1995. Functional documents for computer systems. *Science of Computer Programming* 25, 1 (10 1995), 41–61.

[29] Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. 2010. An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering* 15, 5 (2010), 493–522.

[30] Jesús Sánchez-Cuadrado, Juan De Lara, and Esther Guerra. 2012. Bottom-up meta-modelling: An interactive approach. In *MODELS (Lecture Notes in Computer Science)*, Robert B. France, Jrgen Kazmeier, Ruth Breu, and Colin Atkinson (Eds.), Vol. 7590 LNCS. Springer Berlin Heidelberg, 3–19.

[31] Mafalda Santos, Catarina Gralha, Miguel Goulao, João Araújo, Ana Moreira, and Joao Cambeiro. 2016. What is the impact of bad layout in the understandability of social goal models?. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 206–215.

[32] Bonita Sharif and Jonathan I Maletic. 2010. An eye tracking study on the effects of layout in understanding the role of design patterns. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 1–10.

[33] Lyrene Silva, Ana Moreira, João Araújo, Catarina Gralha, Miguel Goulão, and Vasco Amaral. 2016. Exploring Views for Goal-Oriented Requirements Comprehension. In *Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings 35*. Springer, 149–163.

[34] Harald Störrle. 2011. On the impact of layout quality to understanding UML diagrams. In *VL/HCC, 2011*. IEEE, 135–142.

[35] Robert Tairas, Marjan Mernik, and Jeff Gray. 2008. Using ontologies in the domain analysis of domain-specific languages. In *International Conference on Model Driven Engineering Languages and Systems*. 332–342.

[36] Axel Van Lamsweerde and Emmanuel Letier. 2004. From object orientation to goal orientation: A paradigm shift for requirements engineering. In *Radical Innovations of Software and Systems Engineering in the Future*. Springer, 325–340.

[37] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. 2002. Goal question metric (GQM) approach. *Encyclopedia of Software Engineering* (2002).

[38] Maria Jose Villanueva, Francisco Valverde, and Oscar Pastor. 2014. Involving end-users in the design of a domain-specific language for the genetic domain. In *Information System Development*. Springer, 99–110.

[39] Markus Völter, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, and Wachsmuth. 2013. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform. 558 pages.

[40] Tobias Walter, Fernando Silva Parreiras, and Steffen Staab. 2009. OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In *Model Driven Engineering Languages and Systems SE - 32 (Lecture Notes in Computer Science)*, Andy Schürr and Bran Selic (Eds.), Vol. 5795. Springer Berlin Heidelberg, 408–422.

[41] Fernando Wanderley, Denis Silva da Silveira, João Araujo, and Maria Lencastre. 2012. Generating feature model from creative requirements using model driven design. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*. ACM, 18–25.

Artifacts for Paper #35 of the Software Language Engineering Conference 2017

A Requirements Engineering Approach for Usability-Driven DSL Development

Authors: Same as paper

Abstract: This document presents the artifacts used to evaluate the requirements engineering approach described in paper #35 of the SLE 20017 conference. The artifacts mainly consist of requirements, use case and architecture models of the presented Gyro robot DSL example and the metamodel of the introduced DSSL language. The models can be edited via the Eclipse-based RDALTE tool. The document first explains how to install the tool. Then for every metamodel and model presented in the paper, the document describe how the elements can be viewed and edited using the tool.

1 Overview of Archive

The archive contains this document, the submitted paper and the RDAL language specification document. In order to simplify the evaluation of the artifacts, we have prepared Eclipse packages for the Windows and Linux platforms into which all the required tools have already been installed. Example projects for the models of the paper are provided and can easily be installed in the tool as described in the installation instructions section below.

2 Installation

First install Eclipse with the proper tools and then install the example projects following the instructions below.

2.1 Installing Eclipse

2.1.1 Using the Prepared Packages

Pre-installed Eclipse packages for the Windows and Linux platforms can be downloaded at the following links:

Windows: https://partage.mines-telecom.fr/index.php/s/UTr4TPqkNjv3Wsj/download

Linux: https://partage.mines-telecom.fr/index.php/s/yjiZZZC1nF6Zttw/download

Mac: https://partage.mines-telecom.fr/index.php/s/sC0w2jTGb5BCIyY/download

Unzip the package and run the Eclipse executable.

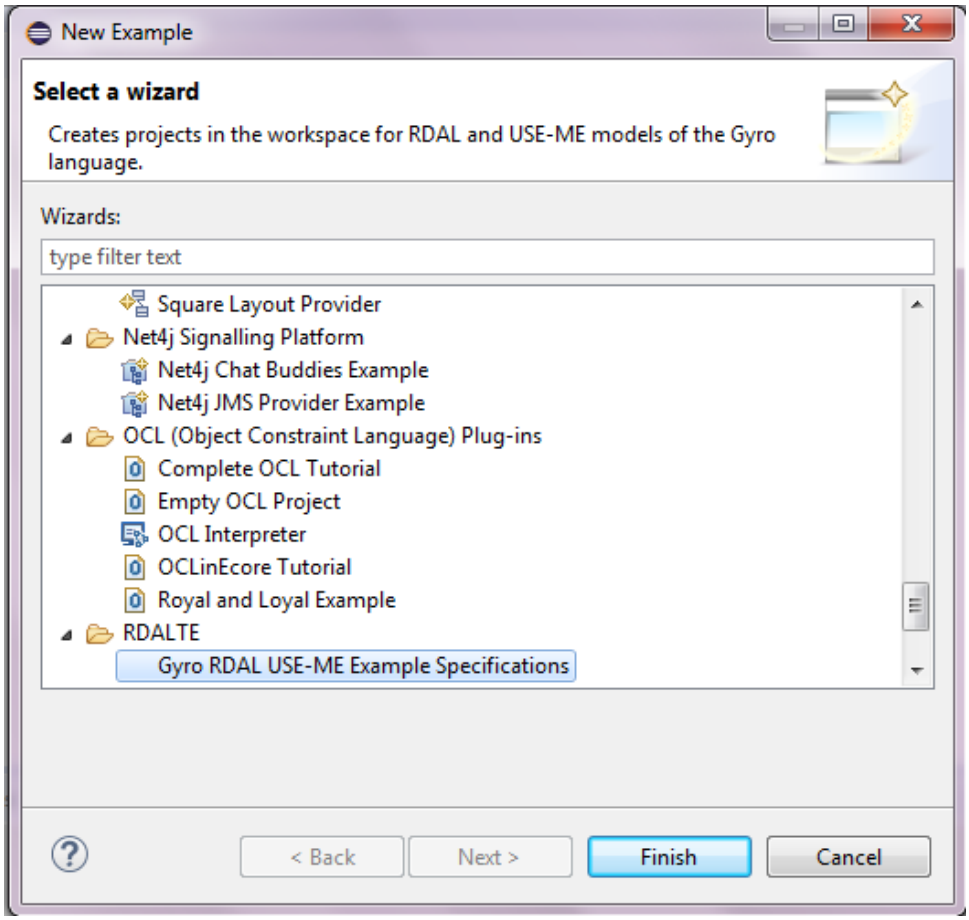
2.1.2 From the Eclipse Modeling Tools Package

Follow the instructions at https://mem4csd.telecom-paristech.fr/blog/index.php/rdal/ in order to install the tools from a fresh Eclipse.

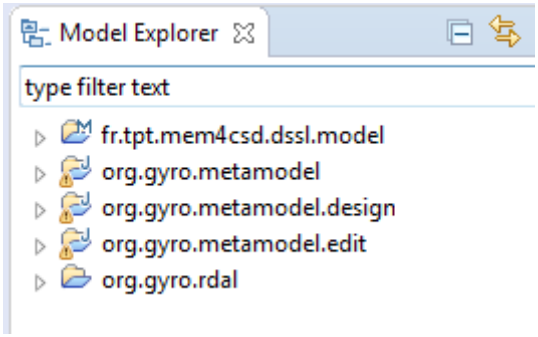
2.2 Installing the Example Projects

Switch to the *Modeling* perspective by opening menu *Window>>Perspective>>Open Perspective>>Other*. Select the *Modeling* perspective from the displayed dialog box.

Next, install the example projects by opening menu *File>>New>>Example*. Select the *Gyro RDAL USE-ME Example Specifications* as shown in the figure below and click *Finish*.



This will install in your workspace the following 5 projects:



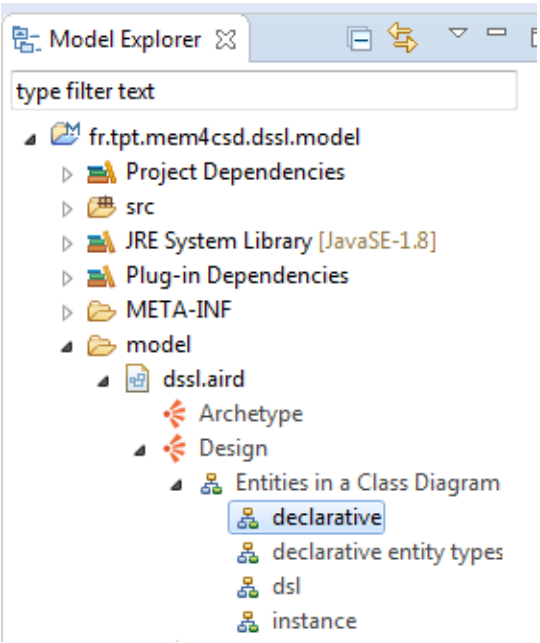
3 Editing and Viewing the Models

The following describes how to view the models presented in the paper. The section or figure number of the paper corresponding to the presented artifacts is written in parenthesis. Editing the models is performed with the provided EMF tree editors for the languages. Double-click the file in the model explorer view to open the file. Select the element of interest in the editor and view its properties via the standard Eclipse *Properties* view. If the properties view is not visible, right click any element in the editor and click *Show Properties View* in the displayed contextual menu to open the view.

Note that the presented tool is an initial prototype that is still being developed. For instance, the concrete syntax for the DSSL and RDAL languages presented in the paper are not yet implemented in the tool. Therefore, the models can only be edited via the EMF tree editors. In addition, the presented models are not complete and mostly contain the elements presented in the paper to illustrate the approach. For example the Ecore metamodel for the Gyro language and its Sirius concrete syntax model are issued from an initial implementation of the language and therefore differ from the real Gyro language.

3.1 DSSL Language (Section 4)

The class diagrams of the DSSL language presented in section 4 of the paper can be viewed by unfolding project *fr.tpt.mem4csd.dssl.model* as shown in the figure below:



Each part of the metamodel presented in the figures of section 4 can be opened by double-clicking the corresponding diagram under the *dssl.aird* file as shown in the figure above for the declarative part.

3.2 Models for the Gyro Language (Section 5.1)

The following presents the models corresponding to the figures of section 5.1 of the paper.

3.2.1 System Overview Diagram (Figure 6)

The DSSL entity types of the system overview diagram can be seen in the *gyro.dssl* file of the *org.gyro.rdal* project, under the root *Specification* element. The RDAL system overview element located under the root RDAL *specification* element can be viewed from the properties view by selecting the element from the *gyro.rdal* file. Note the reference to the DSSL Gyro entity type element from the *System To Be* property.

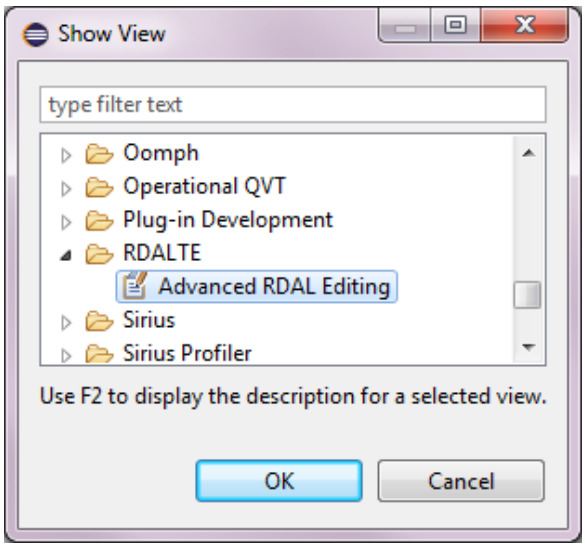
3.2.2 Normal Use Context Diagram (Figure 7)

The DSSL *Normal Use* context specification can be seen in the *gyro.dssl* model. Unfold the element to view the contained entity instances of the types of the system overview and their connections. On

the RDAL model, see the *Normal Use* context element contained in the system overview element and traced to the Gyro DSL entity instance of the DSSL context via the *System To Be* property.

3.2.3 System Goals (Figure 8)

The goals can be seen under the *Gyro System Goals* goals package of the *gyro.rdal* model. RDALTE provides an additional view to edit some elements of a RDAL model. In order to show this view, open menu *Window>>Show View>>Other* and select the *Advanced RDAL Editing* view as shown in the figure below.



The view provides a first tab allowing selecting the element of the DSSL model to which the goal selected in the RDAL file is allocated. The second tab allows selecting the language to be used for expressing the goal (including natural language for high level goals) and to edit and evaluate the defined expression when a formal language is used. Unfold the goal elements in the RDAL model to see their rationales and stakeholders. Also see property *Use Cases* of goal elements for the trace to use cases that achieve the goal (e.g. *G1*).

3.2.4 Use Cases (Figure 9 and 10)

Open the *gyro.jucm* file under the *org.gyro.rdal* project to view the use cases for the Gyro DSL. Double-click the diamonds in the first diagram to navigate to the called sub use cases (e.g. *Create Root*). Note that not all sub use cases have been modeled in this preliminary version. Also displaying a use case diagram automatically switches the perspective to that of the jUCMNav tool so you need to switch back to the *Modeling* perspective after.

3.2.5 Environmental Assumptions (Figure 11)

The environmental assumptions are located under the *Gyro Functional Requirements* requirements package of file *gyro.rdal*. Use the RDAL editing view to see the allocation of assumption to the DSSL entity types and the OCL expressions of the assumptions. Press the *OCL* button to evaluate the OCL constraints. Note the rationale contained by the assumptions and stakeholder association.

3.2.6 Functional Requirements for the Gyro Abstract Syntax (Figure 10)

The *Gyro* DSL entity type is located in the DSSL file and contains an *Ecore abstract syntax* element that itself refers to the actual Ecore package of the Gyro metamodel. This metamodel is located

under the *model* folder of project *org.gyro.metamodel*. On the RDAL model, requirement *R1* is assigned to the *Gyro* DSL entity type and expressed in natural language. *R1* is refined into *R1.1* and *R1.2* through the *requirement refinement* element *R1*. *R1.2* is further decomposed into leaf requirements *R1.2.1* and *R1.2.2*, which are both assigned to the gyro Ecore package and expressed in OCL. Such requirement can be automatically verified by clicking the OCL button of the constraint view. The OCL expressions make use of libraries that can be seen on the *Constraints Libraries* tab. Select a row from the table to open the file. Also note the link from these leaf requirements to steps of the UCM use case diagram through property *Function Used In* in the Eclipse properties view.

3.2.7 Functional Requirements for the Gyro Graphical Concrete Syntax (Figure 12)

The *Gyro* DSL entity type represented in the DSSL model contains a *Sirius Concrete Syntax* element that itself refers to a *Sirius Specification* defining the Gyro graphical syntax. The Sirius model is located under the *description* folder of project *org.gyro.metamodel.design*. Note that the Sirius diagram model is not complete and only the elements required for the provided example requirements are defined.

On the RDAL model, requirement *R1.1* is decomposed into the leaf requirements *R1.1.1.1* and *R1.1.1.2*, which are both assigned to the Sirius *Default* diagram layer providing the visual elements for concepts of the Gyro metamodel. The requirements expressed in OCL can be automatically verified by evaluating the constraints against the assigned design element.

3.3 Models for the USE-ME Language (Section 5.2)

The following presents the models corresponding to the figures of section 5.2 of the paper for the USE-ME language.

3.3.1 Integration between USEME and RDAL-DSSL (Figure 13)

The RDAL and DSSL parts of the figure can be viewed by opening their model files presented above. The USE-ME model can be viewed by opening the *gyro.useme* file located under project *org.gyro.rdal*.

3.3.2 USE-ME Models (Figure 14)

The Gyro goal diagram of the USE-ME language of figure 14 as well as all other USE-ME diagrams can be viewed by opening the diagram in file *representations.aird* of project *org.gyro.rdal*.

Goal modeling diagram is located under the *goalModelling* folder. Detailed explanations on implementation of the Gyro usability model can be found in:

Ankica Barišić, Vasco Amaral, Miguel Goulão, ***Usability Driven DSL development with USE-ME****, Computer Languages, Systems & Structures, 2017, ISSN 1477-8424, http://dx.doi.org/10.1016/j.cl.2017.06.005.*

4 Further Information

4.1 RDAL

See https://mem4csd.telecom-paristech.fr/blog/index.php/rdal/

4.2 USE-ME

See https://github.com/akki55/useme/wiki

Video tutorial https://youtu.be/RjIGFex-zQM

ANNEX VII

Survey Form

In this Annex we include the survey form which was distributed using a Google Forms (*https://docs.google.com/forms* (accessed September 19, 2017)) survey engine.

USE-ME approach survey

This experimental work is conducted within the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) in the context of the evaluation of USE-ME approach which was developed as a part of Ph.D. thesis.

NOVA LINCS is a new unit of the national Science & Technology network in the area of Computer Science and Engineering, launched in 2014/2015, and hosted at the Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa (DI-NOVA), a leading academic department in Portugal.

All information stated as part of this experiment is confidential and will be kept as such.

We would like to emphasize that:
- your participation is entirely voluntary;
- you are free to refuse to answer any question;
- you are free to withdraw at any time.

The experiment will be kept strictly confidential and will be made available only to members of the research team of the study or, in case external quality assessment takes place, to assessors under the same confidentiality conditions. Data collected in this experiment may be part of the final research report, but under no circumstances will your name or any identifying characteristic be included in the report.
In particular, there is no intention of judging you as a person or the skills and experience that you will use in this survey - the goal is the evaluation of the proposed approach!

This questionnaire should not take more than 1h to complete. However, you will be asked to estimate the time you have spent on answering the survey and getting familiar with the USE-ME approach.

It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers. Therefore please take a time to analyze the materials which are shared with you on the following page before continuing to answer the questions.

I would appreciate if you could submit the survey by 07.09.2017.
You can also edit your submitted answers any time till 07.09.2017.

Thank you for taking the time to complete this questionnaire!

Sincerely,
Ankica Barisic, PhD student
a.barisic@campus.fct.unl.pt, Office P3/12
Departamento de Informática - Faculdade de Ciências e Tecnologia / Universidade Nova de Lisboa
Quinta da Torre, P-2829-516, Caparica, Portugal

Prof. Vasco Amaral and Prof. Miguel Goulão are responsible for this experiment and can be contacted at:

- Prof. Vasco Amaral: vasco.amaral@fct.unl.pt; +351 212 948 300 (ext. 10712); Office P2/3
- Prof. Miguel Goulão: mgoul@fct.unl.pt; +351 212 948 536 (ext. 10731); Office P2/17

* Required

1. **Email address** *

Usability Software Engineering - Modelling Environment (USE-ME) approach

Please watch the 15min USE-ME video demonstration before answering to the survey!
Additionally, you are invited to take a look at the article describing the approach and motivation, as well as a tool prototype and example instantiation.

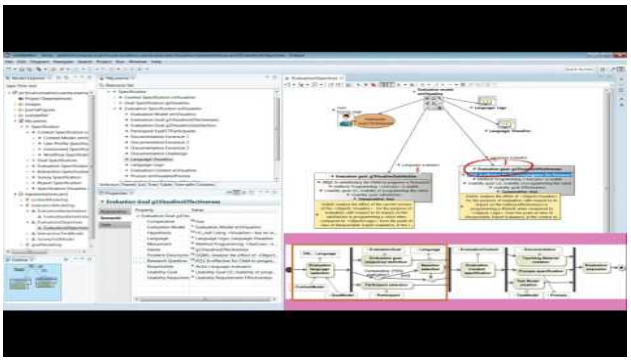
Abstract

The adoption of Domain-Specific Languages (DSLs) is regarded as an approach to reduce the accidental complexity of software systems development. The availability of sophisticated language workbenches facilitates the development of DSLs making them increasingly more popular. This comes at the risk that a badly designed DSL can bring more harm and decrease productivity, when compared to an alternative General-Purpose Language (GPL). In particular, a poorly designed DSL can be too hard to adopt by its domain users. As such, Usability is one of the key characteristics to mitigate this risk as it has an important impact on the achieved productivity of DSL users.

The current state of practice in Software Language Engineering (SLE) neglects the Usability of DSLs. A pertinent research question in SLE is how to systematically engineer Usability into DSLs. We argue that a timely systematic approach based on User Interface experimental evaluation techniques should be used to assess the impact of DSLs during their development process, while the cost of fixing the usability problems is relatively low when compared to fixing them at the end of the development process. For that purpose, the focus of this dissertation is to build a conceptual framework that supports the iterative development process of DSLs concerning the issue of their Usability evaluation.

The Usability Software Engineering Modelling Environment (USE-ME) is a formal approach which is proposed for the usability evaluation of DSLs. A USE-ME prototype is constructed and evaluation process is formally defined in a step by step manner. The resulting modelling instances are intended to be used for decision support when determining the usability of the DSL. A multiple-case study were conducted in order validate the proposed method.

USE-ME approach demonstration video:



http://youtube.com/watch?v=RjIGFex-zQM

USE-ME prototype:

GitHub: https://github.com/akki55/useme
Final Relese: https://zenodo.org/record/345941#.WL1m0BKLRTY
Pilot evaluation materials: https://zenodo.org/record/398830#.WMgHJRKLR24

Journal Article about USE-ME approach:

Barišić Ankica, Vasco Amaral, Miguel Goulão: "Usability Driven DSL development with USE-ME", Computer Languages, Systems and Structures (ComLan), 2017

Pdf URL: https://goo.gl/mCNcQY

Background demographics

The purpose of this section is to collect a basic demographics data about participants. All the information you provide will be kept confidential.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

2. **Full Name:** *

3. **Personal page or a public profile:**

4. **Age:**

5. **Country:**

6. **Degree:** *
   Mark only one oval.
   - PhD
   - MSc
   - BSc
   - Other: _____

7. **Current Work/Research Position:**

8. **Current Work/Research Institution:**

Background experience

The purpose of this section is to collect a relevant experience about participants. All the information you provide will be kept confidential.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

9. **Experience background:** *
   Check all that apply.
   - [ ] Academic
   - [ ] Industry
   - [ ] Other: _____

10. **How many years of working experience do you have?** *

11. **How many years of research experience do you have?** *

_____

12. **How would you rate your level of knowledge related to modelling techniques?** *

Mark only one oval.

- ◯ Expert
- ◯ Advanced
- ◯ Intermediate
- ◯ Emerging
- ◯ No expertize

13. **How would you rate your experience related to Model-Driven Development (MDD)?** *

Mark only one oval.

- ◯ Expert
- ◯ Advanced
- ◯ Intermediate
- ◯ Emerging
- ◯ No expertize

14. **How would you rate your level of knowledge regarding UML?** *

Mark only one oval.

- ◯ Expert
- ◯ Advanced
- ◯ Intermediate
- ◯ Emerging
- ◯ No expertize

15. **How experienced are you with modelling use cases?** *

Mark only one oval.

- ◯ Expert
- ◯ Advanced
- ◯ Intermediate
- ◯ Emerging
- ◯ No expertize

16. **How experienced are you with UML activity or process diagrams?** *
Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

17. **How experienced are you with UML class diagrams?** *
Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

18. **Are you familiar with modelling interaction (communication) diagrams?** *
Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

19. **How would you rate your level of knowledge related to Domain-Specific Languages(DSL)?** *
Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

20. **Did you ever develop a DSL?** *
Mark only one oval.

- Yes, I developed few functional DSLs of which at least one is widely used
- Yes, I developed a functional DSL
- Yes, I developed few prototypes
- Yes, in the context of faculty course
- No

21. **How would you rate your level of knowledge regarding the Eclipse working environment?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

22. **How experienced are you with the Eclipse Modeling Framework (EMF)?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

23. **How experienced are you with the Sirius Modeling tool?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

24. **How experienced are you with software requirements engineering?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

25. **How familiar are you with goal-oriented requirements approaches?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

26. **How experienced are you with agile development?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

27. **How would you rate your level of knowledge regarding Human-Computer Interaction?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

28. **How experienced are you with User-Centered design techniques?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

29. **How familiar are you with empirical experiments?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

30. **How familiar are you with usability testing?** *

Mark only one oval.

- Expert
- Advanced
- Intermediate
- Emerging
- No expertize

31. **Can you describe (refer to) your previous HCI experience?**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

32. **Did you watch the presentation video?** *

Mark only one oval.

- Yes
- No

33. **Did you read the paper presenting the approach?** *

Mark only one oval.

- Yes
- No

34. **Did you try to use the tool?** *

Mark only one oval.

- Yes
- No

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

35. **There is a lack of systematic approach for usability evaluation of DSLs** *

Mark only one oval.

- Strongly agree *Skip to question 37.*
- Agree *Skip to question 37.*
- Don't know *Skip to question 37.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

36. **Can you please let us know why do you disagree with a statement: "There is a lack of systematic approach for usability evaluation of DSLs"**

37. **Can you provide us a reference for any systematic approach for DSLs usability evaluation?**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

38. **Usability evaluations are necessary for a DSL development in practice** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 39.*
- ◯ Agree *Skip to question 39.*
- ◯ Don't know *Skip to question 39.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

39. **Can you please let us know why do you find that usability evaluations are not necessary for a DSL development in practice?**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

40. **Current usability evaluations of DSLs are too expensive and not reusable** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 41.*
- ◯ Agree *Skip to question 41.*
- ◯ Don't know *Skip to question 41.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

41. **Can you please let us know why do you disagree with a statement: "Usability evaluations of DSLs are too expensive and not reusable":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

42. **It is necessary to specify explicitly a context of the DSL when evaluating its usability** *

Mark only one oval.

- Strongly agree *Skip to question 43.*
- Agree *Skip to question 43.*
- Don't know *Skip to question 43.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

43. **Can you please let us know why do you disagree with a statement "It is necessary to specify explicitly a context when evaluating a usability":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

44. **DSL evolution cycle should include usability re-evaluation** *

Mark only one oval.

- Strongly agree *Skip to question 45.*
- Agree *Skip to question 45.*
- Don't know *Skip to question 45.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

45. **Can you please let us know why do you disagree with a statement "DSL evolution cycle should include usability re-evaluation":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

46. **The provided approach supports modelling of the usability evaluation process for DSLs** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 47.*
- ◯ Agree *Skip to question 47.*
- ◯ Don't know *Skip to question 47.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

47. **Can you please let us know why do you disagree with a statement "The provided approach supports modelling of the usability evaluation process for DSLs":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

48. **Concepts modelled by the USE-ME framework are relevant for DSL development** *

Mark only one oval.

- Strongly agree *Skip to question 49.*
- Agree *Skip to question 49.*
- Don't know *Skip to question 49.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

49. **Can you please let us know why do you disagree with a statement "Concepts modelled by the USE-ME framework are relevant for DSL development":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

50. **The USE-ME approach is easy to understand** *

Mark only one oval.

- Strongly agree *Skip to question 51.*
- Agree *Skip to question 51.*
- Don't know *Skip to question 51.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

51. **Can you please explain what did you find difficult, or hard to understand about the USE-ME approach:** *

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

52. **The approach is independent of the particular DSL development approach** *

Mark only one oval.

- Strongly agree *Skip to question 54.*
- Agree *Skip to question 54.*
- Don't know *Skip to question 54.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

53. **Can you please let us know why do you disagree with a statement "The approach is independent of the particular DSL development approach":**

54. **Can you point the DSL development scenario in which you believe that USE-ME approach is hard to be adopted:**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

55. **Approach is suitable to be applied from the early stage of the development of the DSL** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 56.*
- ◯ Agree *Skip to question 56.*
- ◯ Don't know *Skip to question 56.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

56. **Can you please let us know why do you disagree with a statement "Approach is suitable to be applied from the early stage of the development of the DSL":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

57. **DSLs targeting large user groups can benefit from the investment in application of the USE-ME approach iteratively** *

Mark only one oval.

- Strongly agree *Skip to question 58.*
- Agree *Skip to question 58.*
- Don't know *Skip to question 58.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

58. **Can you please let us know why do you disagree with a statement "DSLs targeting large user groups can benefit from the investment in application of the USE-ME approach iteratively":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

59. **The USE-ME tool makes it feasible for a DSL engineer to model a usability evaluation** *

Mark only one oval.

- Strongly agree *Skip to question 60.*
- Agree *Skip to question 60.*
- Don't know *Skip to question 60.*
- Disagree
- Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

60. **Can you please let us know why do you disagree with a statement "The USE-ME tool makes it feasible for a DSL engineer to model a usability evaluation":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

61. **The USE-ME tool is expressive enough for specifying usability evaluation of DSL** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 62.*
- ◯ Agree *Skip to question 62.*
- ◯ Don't know *Skip to question 62.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

62. **Can you please let us know why do you disagree with a statement "The USE-ME tool is expressive enough for specifying usability evaluation of DSL":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

63. **The USE-ME tool supports the integration of usability evaluation approach into development process of the DSL** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 64.*
- ◯ Agree *Skip to question 64.*
- ◯ Don't know *Skip to question 64.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

64. **Can you please let us know why do you disagree with a statement "The USE-ME tool supports the integration of usability evaluation approach into development process of the DSL":**

_____

_____

_____

_____

_____

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

65. **Investment into the development of the USE-ME prototype tool into real product is worthy** *

Mark only one oval.

- ◯ Strongly agree *Skip to question 66.*
- ◯ Agree *Skip to question 66.*
- ◯ Don't know *Skip to question 66.*
- ◯ Disagree
- ◯ Strongly disagree

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

66. **Can you please let us know why do you disagree with a statement "Investment into the development of the USE-ME prototype tool into real product is worthy":**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

67. **Can you please provide your suggestions concerning how to improve the USE-ME tool or approach itself:**

Feedback Questionnaire

The purpose of this section is to collect your opinions about the provided approach. Please note that most of the questions are intended to provide a feedback about the USE-ME methodology (i.e. conceptual framework). When the question is referring to the tool itself, it will be posed explicitly.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

68. **Are you familiar with any other tool which supports usability evaluation?** *

Mark only one oval.

- ◯ Yes
- ◯ No *Skip to question 71.*

Usability evaluation support tools

69. **What are the other tools you are familiar with?** *

70. **The USE-ME is more suitable than alternative for usability evaluation of DSLs** *

Mark only one oval.

- ◯ Strongly agree
- ◯ Agree
- ◯ Don't know
- ◯ Disagree
- ◯ Strongly disagree

71. **The USE-ME is more complete than alternative for usability evaluation of DSLs** *

Mark only one oval.

- ◯ Strongly agree
- ◯ Agree
- ◯ Don't know
- ◯ Disagree
- ◯ Strongly disagree

Survey participants suggestions

We would appreciate if you could provide us a contacts or share the survey [https://goo.gl/forms/VRWLzsFxpWkvMkUw2] with a persons which you find to be interested in giving their opinion about USE-ME approach.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

72. **Can you please suggest us some persons which would be interested in giving their opinion about the USE-ME approach?** *

Thank you for filling in the survey!

Please report the approximate time which you have spent on this evaluation and submit the survey.

* It is not possible to save the survey answers before submitting them. After submission, it is possible to edit and change your answers.

73. **How much time did you spend on getting familiar with approach (watching video, reading the article and/or trying the tool)? (in minutes)** *

74. **How much time did you spend on answering this questionnaire? (in minutes)** *