Pedro Miguel Amado Rodrigues Leonardo

Licenciado em Engenharia Informática

Child Programming: An adequate Domain Specific Language for programming specific robots

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Vasco Miguel Moreira Amaral,
Prof. Auxiliar,
Universidade Nova de Lisboa

Júri:

Presidente:	Doutor Francisco de Moura e Castro Ascenção de Azevedo
Arguente:	Doutor André Leal Santos
Vogal:	Doutor Vasco Miguel Moreira do Amaral

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2013

Child Programming: An adequate Domain Specific Language for programming specific robots

To my father, mother and brother

Acknowledgements

I would like to express my appreciation for all of those who have helped and cooperated with me in the past academic years. I need to highlight that without their support, all of this work would have not be the same.

To Professor Dr. Vasco Amaral, who I have sought for advice and guidance in this dissertation development, I want to share my greatest gratitude. Thank you for your effort in helping me in this chapter of my academic life.

A thankful note of appreciation is given to Artica enterprise group for sharing their knowledge and creativity. Especially for André Almeida and Guilherme Martins, I want to show my appreciation for allowing the use of your robot, Farrusco. I have earned a great perspective in our technological world thanks to you.

To Colégio Campo de Flores and Jardim Escola Natel, I want to show my thankfulness for conceding the opportunity to perform this scientific study.

I want to show my thankfulness to Catarina and Ricardo for lending me their language ArduinoFlow, helping me in this study.

My mother Teresa, father Gabriel and brother Nuno had a tremendous importance for the development of this work. Here, I want to share my deepest thanks for the encouragement that you gave me since the beginning of my life. Nancy as well, I want to thank you for your help, for giving me confidence and being so supportive during my academic chapter.

I would like to thank to my friends André, Cristiano, Fábio and João for accompanying me in this stage my life.

Lastly, I want to show my gratefulness to my family and friends who supported me during this thesis.

Abstract

Due to the limited existence of dedicated robot programming solutions for children (as well as scientific studies), this work presents the design and implementation of a visual domain specific language (DSL), using the Model-Driven Development approach (MDD), for programming robotics and automaton systems with the goal to increase productivity and simplify the software development process. The target audience for this DSL is mostly children with ages starting from 8 years old.

Our work implied to use the typical Software Language Engineering life cycle, starting by an elaborate study of the user's profile, based on work in cognitive sciences, and a Domain analysis. Several visual design paradigms were considered during the design phase of our DSL, and we have focused our studies on the Behavior Trees paradigm, a paradigm intensively used in the gaming industry. Intuitive, simplicity and a small learning curve were the three main concerns considered during the design and development phases.

To help validating the DSL and the proposed approach, we used a concrete robotic product for children built with the Open Source Arduino platform as target domain. The last part of this work was dedicated to study the adequacy of the language design choices, compared to other solutions (including commercial technologies), to the target users with different ages and different cognitive-development stages. We have also studied the benefits of the chosen paradigm to domain experts' proficient on robot programming in different paradigms to determine the possibility to generalize the solution to different user profiles.

Keywords: Domain-specific Language, Model-driven Development, Language Engineering, Behavior Trees, Arduino, Children programming

Resumo

Devido à existência limitada de soluções dedicadas à programação de robôs para crianças (tal como estudos científicos), este trabalho apresenta o desenho e implementação de uma Linguagem de Domínio Específico (LDE) visual, utilizando a abordagem de Desenvolvimento orientado a Modelos, para a programação de robots e sistemas autónomos, com o objetivo de aumentar a produtividade e simplificar o processo de desenvolvimento de *software*, tendo como público-alvo crianças que se insiram num grupo etário a partir dos oito anos.

O nosso trabalho aplica o ciclo de vida característico na Engenharia de Software e Linguagens de Programação, estudando o perfil dos utilizadores alvo, baseado em trabalhos de ciências cognitivas e análise de Domínio. Estudou-se os diversos paradigmas visuais para a fase de desenho da nossa LDE, destacando-se as *Behavior Trees*, sendo este um paradigma bem-sucedido e intensamente utilizado na área dos videojogos. Dado o perfil dos utilizadores, os principais objectivos a manter durante as fases de desenvolvimento e desenho da linguagem são simplicidade e uma curva de aprendizagem reduzida.

De forma a validar a LDE proposta, utilizámos um produto robótico para crianças, construído sobre a plataforma *Open Source* Arduino. A última fase deste trabalho foi dedicada ao estudo das decisões tomadas no desenho da linguagem, comparando-a com outras soluções (incluindo tecnologias comerciais) numa avaliação empírica com os utilizadores alvo. Desta forma, foi possível observar qual o paradigma visual mais adequado para crianças pertencentes a um respectivo grupo etário. Igualmente, analisámos os benefícios deste paradigma com peritos no domínio da robótica, de forma a determinar a solução desenvolvida para diferentes perfis de utilizadores.

Palavras-chave: Linguagem de Domínio Específico, Desenvolvimento orientado a Modelos, Arduino, Behavior Trees, Programação com crianças

Contents

List of Figures

List of Tables

1

Introduction

1.1 General Introduction

Nowadays, we are witnessing a strong presence of increasingly complex system designs in several areas, commonly called cyber physical systems [1] (CPS) that combine both computer systems and electronic elements. These projects, also called embedded systems, can be found in the Aerospace Industry, Telecommunications, Automotive, Chemical, Civil, Energy, Health, Manufacturing, Transportation, Entertainment and Education.

Typically, these systems are intended for different users profiles, from the most sophisticated, deep technology knowledge and programming skills, up to less advanced – adults to children. Systems complexity is increasing and heavily dependent on its actual operating domain. These require domain experts to perform increasingly complex tasks and activities. When developing these kind of systems, it is essential to do a *tradeoff* between usability aspects and adequacy of user control to operate the systems involved and complex tasks they can perform.

Interfaces design, and especially languages suitable to user profiles, has been an answer to achieve this *tradeoff*. However, developing these languages is a recent activity in computer engineering that raises many issues related to the systematic approaches, development tools, re-use of components strategy, validation etc..

Robotics' area is a branch of technology that deals with different electronic components. Each one of those components requires specific programming concepts that will define its job with the surrounding environment. Electronics prototyping field has innumerous technologies such as Arduino[1] and Lego MindStorms[2]. These technologies

[1] http://arduino.cc/
[2] http://mindstorms.lego.com/

seek to relate programming concepts with hardware expertise, in order to create interactive objects or environments. Artists, designers and creative people – which may have no technological background – are the main target users for these platforms. However, given the current state of technology, the user must have some experience with programming concepts, as for example the Arduino language resembles C/C++ languages, and has its own programming tool for compilation and upload code to the Arduino board.

Due to these well-succeeded achievements in electronic engineering field that Arduino brought to the robotic community, various projects have been created within elementary and high school environment. Arduino became an additional working and creative platform in education[2] [3].

Programming concepts along with Robotic ideas gave benefits in children learning process, mainly due to interactive aspects since students have the opportunity to observe the code that they develop, such as a simple blink of an LED on the Arduino board.

1.2 Problem Description and Motivation

The current state of the art in programming technologies of rover robots for children is still greatly excluding the younger ages because it is still hard for them to program in a textual language with a complex syntax full of programming languages concepts.

However, to design an appropriate domain specific language 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 pertinent question it arises is if it is possible to build a dedicated DSL, that removes the programming details, to control a rover robot, and at the same time allows children to easily get acquainted with it and have still the possibility to program complex behavior?

The objective of this work is to study and implement, in the domain of Robotics applications, the adequate language constructs and metaphores for a DSL designed for children. Since the DSL is intended to be used in an education environment (primary and secondary), visual elements and textual concepts should be appropriate to the target users. Low learning curve is a mandatory requirement such as productivity increment in creating new programs.

Children should be able to program behaviors for a specific robot, considering its consisting components. They should combine the information from sensing components with the actuators like motors or LED lightning. The DSL should incorporate robot's component functions in order to be understandable and ease of use by children. Thus, the problem covers the development of a suitable DSL for children for education and entertainment areas.

1.3 Context

The DSL development counted with a Robot consisting with common components and functions in the robotic area. The work is being developed under a collaboration between the group ASE CITI/Departamento de Informática and Artica, a company that specializes in development of robotic and audio-visual solutions. Artica provides the Robot and case studies with children that will help evaluate the DSL usability. *Farrusco* is the name of the provided robot which is composed by an Arduino board – attached to this board, there is a set of sensors and actuators components. This robot own a typically physical configuration for the components set, although it is possible to change it. *Farrusco*'s consist on the following components:

- An Infra-red Distance sensor;
- Two collision detectors sensors, named *bumpers*;
- Two motors for each wheel;
- A motor named Servo, that actuates under the Distance Sensor;
- A simple LED over the Arduino board.

The Arduino board is showed in figure 1.1, with its *pins* for sensors and actuators connections.

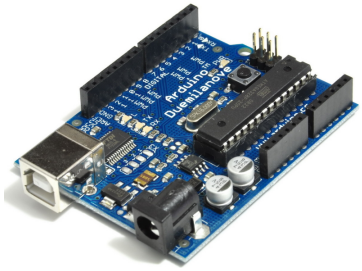


Figure 1.1: Arduino Board

Farrusco, being the chosen robot for the development of this work, is directed to persons with low levels of knowledge in programming and electronic areas. Thus, when designing a simple behavior for Farrusco, e.g. activate both wheel motors so it can move forward, it is necessary some time and effort taken by users. The time spent in learning robotic and programming concepts could also add low levels of motivation for the user.

Children may find difficulties when programming robot behaviors. *Farrusco*'s behaviors are implemented in Arduino textual code – children and adults with limited technological knowledge may encounter problems such as syntax language and logical errors. Developing a DSL for this purpose may help the interaction between the user and the robot. The target user could identify Farrusco and robotic concepts more easily, since the DSL is based on domain specific concepts and ideas. Simple robot behaviors should be created almost instantly, given that the DSL should have a low learning curve.

1.4 The solution

In order to develop a suitable solution, it was necessary to study and analyze the current domain of rover robots for children. This analysis encompasses the study of different user profiles that represent possible target users for the developed DSL. The robot domain had to be limited due to its extensive area and the diversity of concepts and technologies it contains.

The different types of visual programming paradigms were analyzed as well, so we could properly adapt the DSL's usability to the user profiles and the specific robotic domain. Behaviors trees, dataflow and workflow diagrams were the visual paradigms considered in the development of the DSL.

According to the domain experts point of view, we decided to study and analyze the behavior tree paradigm as a visual language, since it has a great success in other domain areas (e.g, coordinating autonomous agents in video games), addressing its applicability to younger age groups. In this way, we access what could be the best paradigm for programming robots, both for children of different ages and at the same time determine the increasing productivity of using such paradigm with the domain experts.

Through a Model-Driven develoment (MDD) approach, it was possible to formalize and create the DSL metamodel covering the specific details from the domain analysis. It also provides an agile development considering possible DSL improvements made along the research. Eclipse workbench and Eugenia tool were the technologies used for the DSL development.

1.5 Language Validation

A great effort was invested in planning and organizing the empirical studies required to evaluate the design of the language. Namely, we looked at the visual paradigms, and design constructs of the language.

As we will describe in the next chapters, we achieved a set of conclusions and learnt lessons that serve as input for the design of a future new evolution of the proposed language in this document.

1.6 Dissertation organization

The dissertation is organized in ten sections:

On chapter 2 the state of the art of this project is analyzed. It includes the themes of Games Engines, Augmented Reality, Visual Programming Languages, Modeling Languages and DSLs.

- Chapter 2 discusses software languages, particularly the Domain-specific languages engineering. Model-driven development is described in this section. This chapter

also compares languages and robotic tools for children, highlighting important aspects for each technology;

- On chapter 3 is presented the state of the art, which shows technologies involving the robotic area, and programming tools for children;
- Chapter 4 shows the domain analysis and the proposed solution for the DSL development. It also describes the user profile characteristics;
- Chapter 5 presents the language design details.
- On Chapter 6 are presented implementation details and the process taken for the DSL development;
- Chapter 7 introduces the evaluation process for the DSL;
- Chapter 8 describes the results taken from the case studies presented in the previous chapter;
- Chapter 9 contains the language evaluation and validation with Arduino domain experts;
- Chapter 10 includes conclusions aspects, contributions and future work for this thesis.

2

Software Languages

In recent years, there has been a growing number of languages used for creating and drawing software, as the interest in engineering languages.

Likewise, the use and creation of DSLs have become increasingly dominant. DSLs focus and describe certain aspects of a system or a particular software fraction. Modeling languages have become very important in model-driven development context. The use of models and metamodels is part of modeling languages. Without the existence of multiple programming and modeling languages, certainly the MDD[1] approach would not have much relevance.

There are many similarities between modeling and programming languages such as:

- Both are used to describe software;
- After transformed and/or compiled, both have to be executed.

2.1 Modeling Languages

Domain specific models are designed to increase the abstraction level that programming languages provide, using concepts that identify the domain of a given problem [4]. With this approach, it is possible to generate final applications produced using a programming language. These applications correspond to the model representing the abstract form of the problem domain. End applications' generation process is usually supported by a *framework* or *API* that uses a specific domain generator [4].

A modeling language can be used to express knowledge or systems in a specific structure, defined by a consistent set of rules. The language can be either graphical or textual.

[1] Model-driven development

A language is composed by a syntactic notation, which may contain an infinite number of elements. This corresponds to a language's abstract syntax, which defines the concepts and how they can relate with each other. The concrete syntax defines the concepts' presentation, together with the semantics that gives meaning to those concepts [5].

2.2 Model Driven Development

Model-Driven development approach, also known as Model-Driven architecture, is a technique or a set of methods for software development, that is a trademark of Object Management Group in the scope of the Model Driven Architecture[2]. The modeling concepts refer to domain concepts instead of mapped into broad technological notions, so full potential of Model Driven Development can be achieved. Models are essential artifacts that describe specific concepts for a target domain. Those contain abstract representations of the knowledge that manage a certain domain [6]. Through MDD, a system is defined by a model that is in conformity with a metamodel.

A model contains problems and concepts that vary by domain. Distinct domains require different languages. DSLs came to solve this problem, allowing the use of domain concepts on the models' specification [7].

2.3 Domain Specific Languages

Domain Specific Languages have the same characteristics as any other language composed by three essential elements:

- Abstract syntax describes the language structure, with its properties, rules and relations;
- Concrete syntax explains the language notations, i.e., it describes how language's elements are represented (visual, textual or a mixed);
- Language semantics defines the concepts' meaning described in abstract syntax.

A domain is limited by an area of interest and knowledge, characterized by a group of concepts and terms understood by persons of that field. Domain analysis is a fundamental piece for a DSL development. Without the domain concepts clear, it is not possible to understand which are the important requirements needed to implement and design the DSL. Domain experts give important information so it is possible to express problems and concepts at a higher level of abstraction. This can be achieved trough a careful analysis of the domain's features. A Feature model can represent those domain concepts and features in a form of a tree. It describes how each concept relates to another.

[2]http://www.omg.org/mda/

The conceptual distance between the problem area and the language used is reduced. The DSL presents concepts that are familiar to the target users within their working domain, instead of general computer terms [8]. The DSL is defined through concepts and features specific to the problem domain, which improve the usability and comprehensibility for the end-users. The main goal for any DSL is its expressiveness power on its target domain [9]. Solution details are hidden from the user, and domain experts do not need to worry about those specific details. This dramatically reduces the development time for a target solution.

Domain specific modeling relates the final application characteristics with domain concepts[6]. However, this technique is not suitable when the application domain is not entirely known.

2.4 DSLs Development Methodologies

As previously said, domain analysis is one of most important factors for starting to develop a DSL. This section describes common methodologies used in DSLs development.

2.4.1 Modeling and Metamodeling

Modeling is used for designing systems, making their comprehension easier. At the same time, they also specify the required functionalities for the target system. Models are used for designing purposes, this happens when the problem domain requirements are too complex to represent in textual code. Thus, models are used to add an higher abstraction layer and hide code implementation.

Metamodeling is a formalism to specify software languages [10]. Metamodeling is the gathering of a collection of concepts within a certain domain. A metamodel is a model used to specify a language [10]. Abstract syntax is defined through a metamodel. It describes language's rules, properties and relations [10]. This also applies to the concrete syntax of the language. Any model that specifies language's details and concepts is considered as a metamodel.

The model is an abstraction of the actual real world concept. Models are commonly represented with elements, their connections and special symbols. Modeling provides: a cleaner architecture presentation; conceptual simplicity; efficiency implementation; scalability and flexibility [7].

2.4.2 Model transformation

A model transformation is a function from abstract syntax models I1, .., In to abstract syntax models O1, .., Om [10]. These transformations are defined in the syntax structure metamodels. This could be achieved through transformation languages, e.g., ATL[3],

[3]http://www.eclipse.org/atl/

Stratego[4]. Through rules and mapping functions defined in the elements from the original model, it is possible to create a new model based on the old one.

These languages are used in the relationship between the domain metamodel and the framework metamodel, used to generate code. A model's instance could be transformed to a verification model that evaluates the instance model's properties.

2.4.3 Code generation

Code generation can be regarded as a form of semantics [10]. The target language, usually has a lower abstraction level than the original. This requires the construction of a code generator. There are several tools designed for this purpose, such as JET[5], Xpand[6], EGL [7]. A code generator parses the instance of the model as a tree, and generates the corresponding code blocks. The generator is implemented and prepared to translate the model to the chosen programming language code.

2.4.4 DSL usability evaluation

DSLs are a way to increase productivity, using concepts of the problem domain. Typically, DSLs target users do not need a deep knowledge in programming languages concepts. DSLs came to fill the gap between domain experts and computational solutions platforms [11]. Human/Computer Interaction presents the same objectives as DSLs. Both should increase the users' efficiency, while performing their duties without having to cause extra organizational costs [11]. The language engineer must get involved in the target domain, so he can study and analyze specific problems, concepts and terms. At the same time, the engineer developer shall have the ability to build the target language [11].

The evaluation of User Interfaces is related with a qualitative software characteristic named Usability. It is defined by quality standards in terms of achieving the Quality in Use[8]. Usability is defined 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" [12][9]. There are forms to evaluate usability such as:

- Formally through some analysis techniques, taking models and simulations, e.g measuring the elapsed time to complete a given task.
- Automatically by a computerized procedure. This is possible when application prototypes are available.
- Empirically, through experiences where users test the application. This technique is recommended at all stages of development;

[4]http://strategoxt.org/
[5]http://www.eclipse.org/modeling/m2t/?project=jet
[6]http://www.eclipse.org/modeling/m2t/?project=xpand
[7]http://www.eclipse.org/proposals/egl/
[8]ISO/IEC 9126 Quality Standards. 2004. http://www.iso.org/iso/
[9]ISO 9241-11

- Heuristically, auto evaluating the product, based on self-judgments.

Building a new DSL, three main design objectives are required[11]:

- The language should capture the domain expressivity;
- It must ensure compliance with the standards of a given domain;
- It should overcome previously identified problems in the domain.

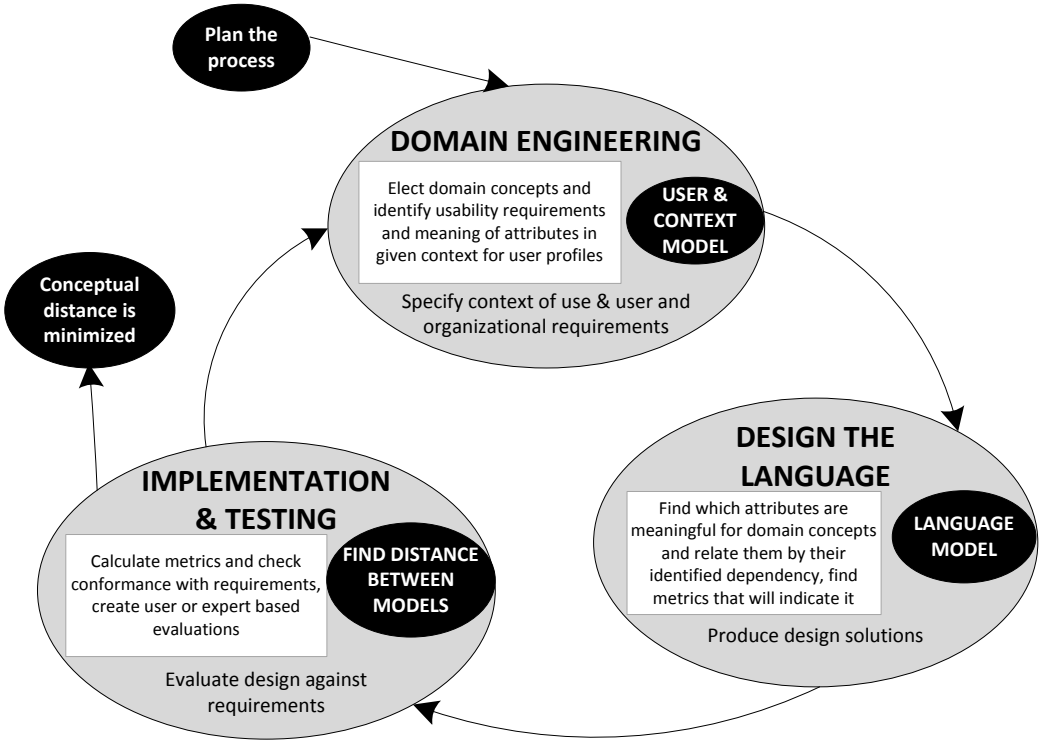


Figure 2.1: DSLs' usability evaluation process [11]

As proposed in [11], the usability evaluation should be done during the DSL development. According to this proposal, usability requirements should be gathered/identified during *Domain Engineering stage* and at the same time, domain concepts are collected. During the *Design The Language* stage, each domain concept should be identified as its relevance within the specific domain. In *Implementation and Test stage*, the language engineer should validate the usability requirements' list. Each stage is depicted in figure 2.1.

It is important to measure during the language development, the distance between the language model and the user context model, through predefined metrics. The smaller the conceptual distance the higher the level of Quality in Use [11].

2.4.5 Languages metamodeling Workbench

Metamodeling tools allow a simplified development process of a DSL. These tools offer features to develop the language metamodel, establishing relations, properties and rules to the models. Visual editor is created with these tools as well, so end-users could produce new models and generate the corresponding code. There is a wide choice of tools for the DSL development, such as *Eclipse Modeling Framework* [10] that uses *Graphical Modeling Framework* [11] to design models. Microsoft has one DSL development tool, named *Visual Studio Visualization and Modeling SDK* (the old DSL Tools)[12]. It features the same general functionalities as *Graphical Modeling Framework*.

Eugenia[13] is a tool for Eclipse Workbench that generates automatically the visual editor (*GMF*) and its corresponding concepts, through a single annotated metamodel, the *Ecore* file.

2.4.5.1 Workbench selection

The chosen metamodeling workbench for the DSL implementation was *Eugenia* for *Eclipse*. This tool simplifies the generation process of models to implements a GMF editor [13]. It offers an extensive list of functionalities for developing the DSL. Creating a new GMF editor from scratch brings some challenging aspects, since it has many configurable details. Eugenia provides a higher-level abstraction, hiding the complexity of GMF [6]. These characteristics allow a rapid prototyping phase, so users can be involved earlier in order to achieve their feedback.

2.5 Visual Languages Paradigms

A visual programming language allows the user to implement programs, through the manipulation of visual elements or objects. This manipulation is performed in a visual way, allowing the user to understand quickly programming mechanisms, increasing their accessibility to new systems [14]. Users with no programming background may implement programs in a simpler form.

A visual programming language is represented by a meta-model, as opposed to textual programming languages. These textual languages representation is made through grammars practices [10]. Visual languages are classified according to the expression type used. They can use expressions based on forms, icons, diagrams or even the combination of these techniques. Figure 2.2 depicts visual programming language paradigms, each one with their own type of visual expression.

[10] http://www.eclipse.org/modeling/emf/
[11] http://www.eclipse.org/modeling/gmp/
[12] http://archive.msdn.microsoft.com/vsvmsdk
[13] http://www.eclipse.org/epsilon/doc/eugenia/

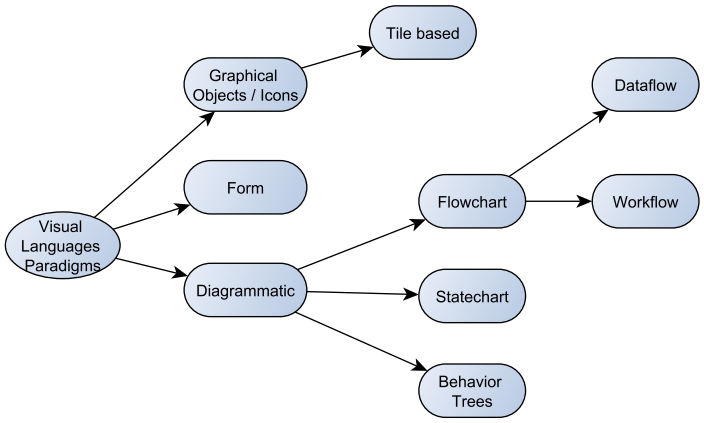


Figure 2.2: Visual Programming Languages paradigms

Iconic / Graphical Elements based VPLs

The visual programming languages based on icons or graphical elements, hold figures (icons) denoting objects in the program with their own corresponding function or real world representation [15]. Tile-based languages are represented by graphical tiles (elements) which can be manipulated by the programmer as building blocks [16]. Typically this type of VPLs [14] programs are created by building a vertical sequence of tiles, maintaining a sequential execution from top to bottom. They also present spatial concerns, since each tile is directly connected to its corresponding attached tile – there are no edges or arrows to describe icons/nodes relationships. The following chapter shows several languages which apply this visual programming paradigm such as Scratch programming language. Icons and visual artifacts are usually combined with other language paradigms, since they are able to represent specific problem domain aspects [14].

Form based VPLs

Form-based paradigm is similar to a spreadsheet, in which the form is typically shaped as a grid and its contents are represented by cells. Forms corresponds to an abstraction of conventional paper forms, which facilitates non-experts users to organize data into a structured representation (e.g, tables) [15]. The user programs by creating a form and specifying its contents, commonly seen in commercial spreadsheets [14]

Diagrammatic VPLs

Diagrammatic VPLs are based on nodes (which may represent program states or functions), and edges (corresponding to transitions or arrows which symbolize a flow of work). We may characterize the diagrammatic VPLs in other three sub categories – Flowcharts, StateCharts and Behavior Trees. These are interesting to study since they

[14] Visual Programming Language

can represent states, work flows or even data flow processing, relevant to describe autonomous agents programs/behaviors. Flowcharts depicts algorithms or processes, through the step boxes ordered by arrows which arrange the program structure and its corresponding flow – activities and decisions are expressed through this paradigm nodes.

Dataflow

Dataflow paradigm provides a view of computation which shows the data flowing from one filter function to another through the representation of arcs [17]. In the *Dataflow* execution model, a program is represented by a directed graph where the nodes of the graph represent primitive instructions (e.g arithmetic or comparisons functions). Directed arcs between the nodes represent the data dependencies between the instructions [18]. This programming paradigm allows the representation of control-flow or data structures execution.

Labview[15] language and Openwire library [16] are software solutions which apply the dataflow paradigm in their programs.

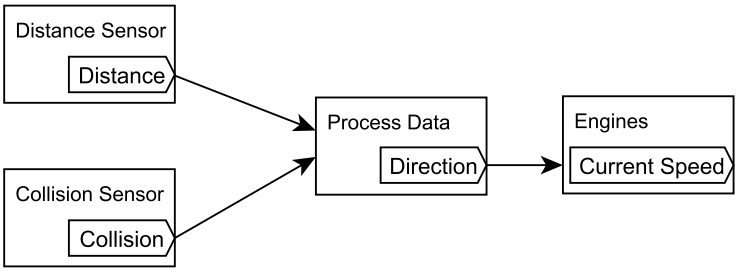


Figure 2.3: Dataflow paradigm example, describing a robot behavior

WorkFlow

Workflows are a similar way to express information-processing scenarios. A Workflow is defined by nodes or blocks that represent data processes and connections, which characterizes the link between those nodes. The links represent the relationships that apply constraints on the execution model. Figure 2.4 shows an example of a *workflow* program. Typically, Workflow models represent operations or sequence of operations which may correspond to a single entity (e.g. an electronic component). This is the main difference between Workflow and Dataflow paradigms. Dataflow paradigm define operations with input and outputs [17] clearly separated – this paradigm defines the structure of a system through the components which composed it. The Workflow paradigm declares the sequence of operations for a single entity/component.

[15]http://www.ni.com/labview/
[16]http://www.mitov.com/products/openwire

YAWL[17] is an example of a workflow system, based on a modeling language which uses the Workflow paradigm to handle transformations, resourcing requirements and control-flow dependencies.

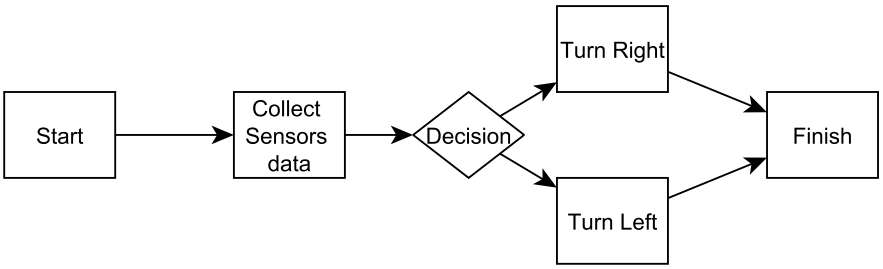


Figure 2.4: Workflow paradigm example, describing a robot behavior

Statechart

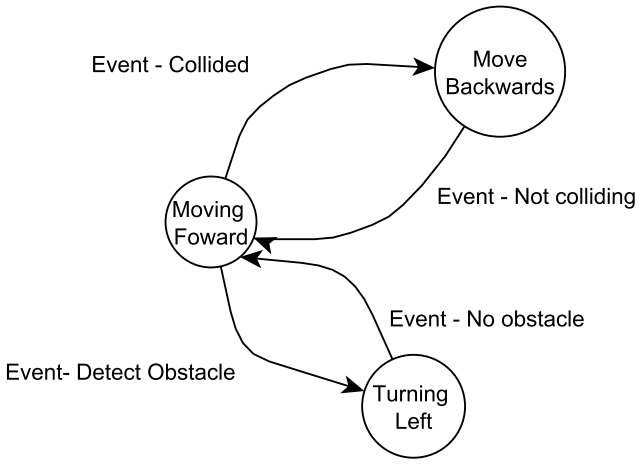


Figure 2.5: Statechart paradigm example, describing a robot behavior

Statecharts is another diagrammatic visual programming paradigm based on finite state automata. The states diagrams are used to represent behaviors of a system, describing the current state of the system itself, and the arrows define the transitions/behaviors between states. Figure 2.5 depicts an example of this visual programming paradigm, characterizing a robot behavior; in this particular case, its own speed and direction.

Statechart [19] visual language uses the conventional state diagrams to specify software requirements.

[17] http://www.yawlfoundation.org/

Behavior Trees

A different way to represent a visual language may be through *Behavior Trees*. These act as an alternative to state machines and are composed by an hierarchy of behaviors, with an objective to fulfill. Each node may have a specification that determines how the actions of its children will be executed, which may be in parallel or sequentially. Children return its status to the parent node, and this successively until the root of the tree [9]. A behavior tree is a structure containing behaviors organized in a tree. It is composed by a complex behavior which is mapped into smaller simple behaviors through its branches [20]. This descending order of complexity provide a structured manner to define complex behaviors through simple tasks hierarchically defined. The behavior tree concept is commonly used to encode game artificial intelligence agents [21] by gaming and artificial intelligence domain experts, in a modular, scalable and reusable manner [22] – valuable characteristics in a DSL.

Figure 2.6 shows a simple robot behavior through the *Behavior Tree* paradigm. It contains two control nodes: Parallel and Decide nodes. These two nodes control the *behavior tree's* flow and execution states. Parallel node simply runs all of its children at the same time. The decide node decides which children should run at a specific time and situation. Leaf nodes represent the most primitive actions that could be taken by an agent.

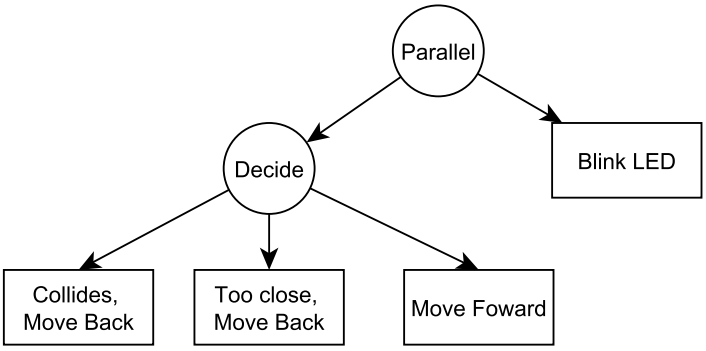


Figure 2.6: Behavior Tree paradigm example, describing a robot behavior

In this particular case, the Parallel node activates the robot's LED, and executes the Decide node as well. The decide node executes always the first child node, which is typically a condition node. If the condition succeeds, the decide node proceed in a sequential order to the following children. So, if the robot collides with an obstacle, it moves back until it is in a safe position. Once this reaction to the obstacle is finished, it proceeds with the normal behavior – moving forward. Thus, it is possible to define a series of behaviors through the showed paradigm.

Visual Programming Language remarks

There are significant differences between visual programming language concepts and textual languages. It is, therefore, a different way of programming, whether for an experienced programmer or a person who has no background/programming knowledge. The programmer can focus the solution through visual elements which handle lower information density. However, this type of programming languages also comprise negative aspects – e.g., the language's implementation area/surface tends to increase over complex solutions, making its analysis and reading difficult to achieve. *Containers* are blocks that group several developed elements of code. Conversely, they also prevent the occurrence of syntax errors, a fairly common problem in textual languages and the automatic parallelization of code, an important characteristic which is one of the most difficult challenges presently [18]. Empirical studies also demonstrated that visual languages are more intuitive and quick to understand when compared to textual languages [23].

2.6 Summary

In this chapter we analyzed Modeling and Domain-Specific Languages along with the Model Driven approach. We also showed typical methodologies and the development process to implement a new DSL. It is important to highlight the iterative process within the DSL implementation and design, in order to evaluate its usability and identify possible language flaws or domain concepts that could mislead the target user. The domain analysis provides the essential specific concepts and knowledge necessary to support the language design.

Visual Programming Languages paradigms were also studied, considering that the language visual abstraction is an important design decision that could easily affect its usability and final results with the target users. For the examined paradigms, we took a special effort for the behavior tree – it has the advantage to be a valuable paradigm and structure to implement autonomous agents behavior through the definition of a tree as previously mentioned. It is typically employed by expert users in the artificial intelligence domain. It could also be interesting to introduce this concept to younger learners, knowing that the behavior design is always abstract – what is the best approach to construct and organize robot behaviors? We form the hypothesis that the behavior tree paradigm is adequate to program robot behaviors by children, since we found no published works to contradict this assumption. Since this paradigm shows a great success for behaviors implementation, we will rely and study this particular aspect for younger users. And as we observe in the following chapters, Piaget says that children can only use abstractions to represent knowledge until they are eleven years old, we need to observe this aspect in real study cases.

3

Entertainment and Education Technologies for Children

Children have a desire to explore and experience the physical and conceptual environment [24]. Several technologies have been developed in the areas of Computer Science and Robotics, in order to aid children's development. Obviously we cannot disregard the works, applications and languages already made in this domain, so we can take it as an advantage in order to study its features, limitations and visual paradigms and further compare those with our DSL. Visual languages introduce programming concepts to children[25], while Robots perform the developed code in the real world. Children have the opportunity to observe their own developed program, running in a physical robot. The following sections demonstrate languages and current technologies for children, presenting their goals and objectives.

3.1 Lego MindStorms

Lego MindStorms is perhaps the most known technology in Educational robotics, having children as target users. Several case studies [26] have been done about this technology in order to understand which are the main individual needs of children, when working with robots. *Lego MindStorms*' approach is to combine the concepts of hardware and software, so the users could develop and deploy programs containing behaviors which shall be executed by the robot. The robot is built by children, so they could understand connections mechanics and each component's function, such as sensors and actuators.

Lego's visual programming language is called *RobotLab Labview*, presenting blocks as elements to build a program. Each block represents a programming concept, such

as execution control element, e.g loops, conditions, arithmetics, 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. Each block may contain options or arguments to trigger a specific action, e.g. the sound block shows a list of possible sounds that can be played by the robot. Figure 3.1 shows an example made with this programming language.

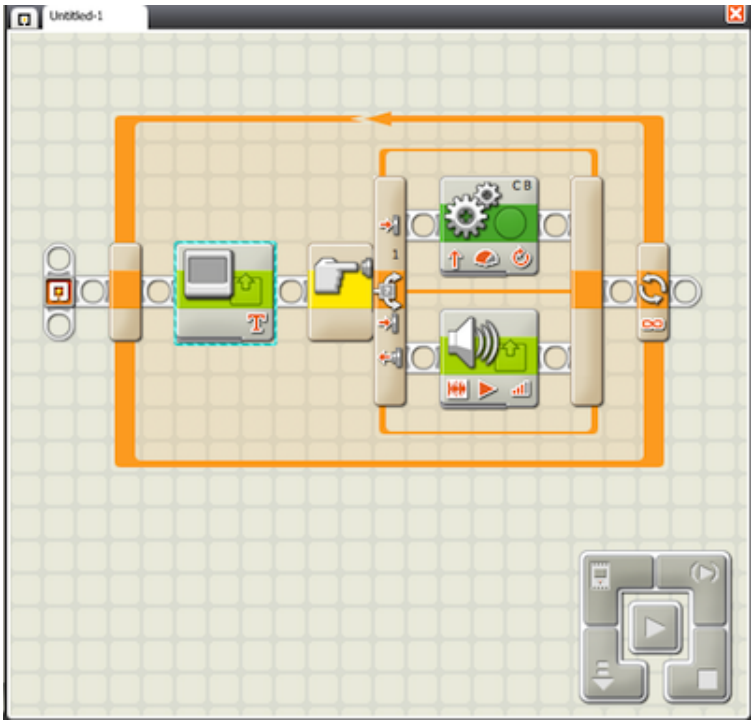


Figure 3.1: Lego MindStorms' Programming Language

These robots contain negative factors, like the constant need for recalibration of its sensors and motors. Lego MindStorms' Tool contains a function to solve this problem. This is one important feature to identify, since calibration is a common problem in the robotic world. The visual language brings appealing icons and figures, but the development of complex behaviors may become difficult [27]. The increasing number of blocks and the size of the flowchart becomes difficult to analyze and read the program solution. Thus, non-trivial behaviors are difficult to implement in a visual programming language like this.

3.2 Scratch

Scratch[1] is a pedagogic programming tool that has children as target users. The *Scratch* grammar is based on a collection of graphical "programming blocks", and it aims to teach basic programming concepts in a simpler and creative way. It proposes an easy to

[1]http://scratch.mit.edu/

learn approach, through visual elements. In order to avoid syntax errors, *Scratch* has no ambiguous syntax or punctuation of traditional textual programming languages, which could lead to user's frustration and consequent leave of the language [28] [29]. *Scratch*'s blocks are shaped to fit together only in ways that make syntactic sense. Those blocks represent programming concepts, such as control loop structures (forever, repeat) or conditionals. Each type of block suggests how it connects to a different block:

- *Loops* control blocks are C-shapped suggesting that blocks should be placed inside.
- Blocks that return values are shaped according to the types they return (oval for numbers, hexagons for booleans).
- Conditional blocks have a hexagon-shaped spaces which indicates a boolean is required.

Concepts like parallelization are presented abstractly – launching two sequences of blocks at the same time creates two independent threads that execute in parallel. Scratch maintains the concepts of basic programming given that its goal is to introduce children the programming concepts and logical reasoning. Figure 3.2 shows a simple script on *Scratch* workbench. Whenever the mouse pointer goes over the game character, it should move 5 steps, followed by a pause of 5 seconds.

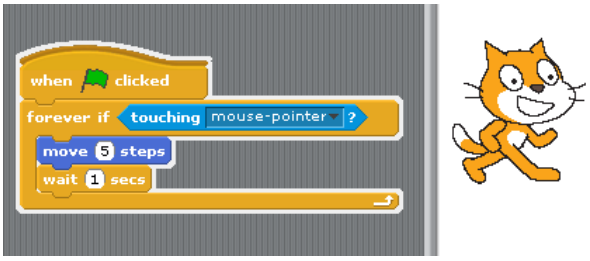


Figure 3.2: Sample *Scratch* script

Implementing games or programs using a visual interface with objects and personal elements, such as photos, icons or figures, increase children motivation and willingness to explore, since the user works on personally-meaningful projects [28]. Thus, the child can create a graphical program, enabling the creation of multiple animations, games and interactive stories. *Scratch* placed high priorities in personalization and diversity so children with widely varying interests could work on projects that they care deeply about. Therefore, project customization is one of the most important features of *Scratch* [28].

3.3 Visual Programming Languages for Arduino

This section will cover the existing visual programming languages for Arduino. All of these languages have a commom purpose: teach users to build projects for Arduino. They encompass an imperative paradigm, with an abstraction level similar to *Arduino's* C++ language.

3.3.1 Scratch for Arduino

Scratch for Arduino[2] combines Scratch visual programming language, presented previously, with Arduino's concepts and features. This tool provides a graphical interface to produce projects for Arduino's interactive objects, e.g. robots, instead of *Scratch*'s game character.

Scratch original blocks were adapted to Arduino main functions, such as write and reading for Analog and Digital pins belonging to Arduino board. Figure 3.3 shows a simple project with a sequence of blocks containing Arduino functions, such as pins voltage control and conditionals blocks together with Arduino sensors information.

This technology has a developed firmware that should be compiled and uploaded to the board before developing a solution. Without this firmware, developed projects are not possible to communicate with Arduino, since the firmware works as a bridge between the computer which has installed *Scratch for Arduino*. Arduino board does not process any data, it simply actuates motors and delivers sensors information to its attached server/PC. This server processes all the information and decides which Arduino's motors should activate. If the connection between the devices is made by a physical cable (the link is commonly maintained by USB cable), the robot will only operate within the cable range.

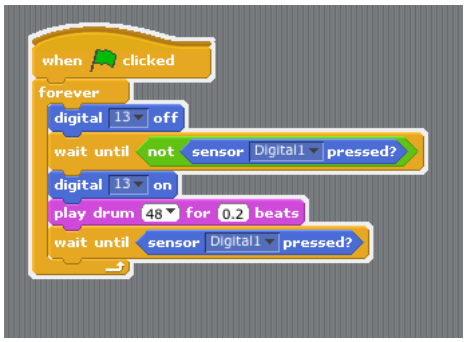


Figure 3.3: *Scratch for Arduino* sample script

Users' developed projects are not compiled or uploaded to the board, making it impossible to maintain the behavior when the connection between the two devices is canceled. In order to maintain the execution of the developed project, the connection between the Arduino and the attached server, must remain plugged.

3.3.2 ModKit

Modkit[3] technology is a similar tool with *Scratch for Arduino*, presented previously. Consequently, it is also based in *Scratch* visual programming language. It has specific blocks for programming Arduino microcontrollers. Bringing programming concepts to the real

[2]http://seaside.citilab.eu/scratch/arduino
[3]http://www.modk.it/

world is the main goal for this technology, combining robotic concepts with programming logical thinking [30]. Children as the target audience for *Modkit*, can develop, compile and upload projects to Arduino board, being able to observe the developed behavior in the real world platform.

Blocks' shape and the workflow programming are similar to *Scratch*. These blocks have a nomenclature analogous to Arduino's commands and functions. *Modkit* followed this approach, so that users with earlier Arduino knowledge could be able to fit in quickly to *Modkit*. Users with no Arduino knowledge would be able to understand these concepts, preparing them to program in the original Arduino textual language [30].

Blocks' shapes suggest the way they fit with each other as previously mentioned languages. This tool runs over any Web browser, allowing automatic updates without users' configuration. An example of this language and tool is shown in figure 3.4.

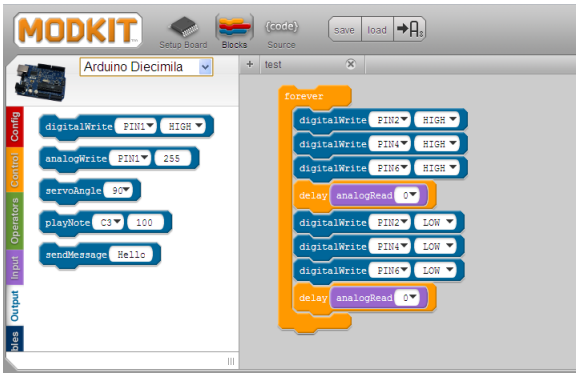


Figure 3.4: *Modkit* sample script

This tool allows to develop Arduino and Arduino compatible software using: graphical blocks, traditionally text code. The graphical developed program is automatically translated to text-based code, so users can observe what the block code would look like in textual form. Therefore, code analyses shall be easier for experienced users.

Modkit specifies Arduino platform as its target domain. Arduino's world is very broad, containing too many concepts and robotic solutions, making *Modkit* a general solution for visual programing projects.

3.3.3 Amici

As opposed to previous technologies, Amici[4] has its own graphical interface elements and a new visual way to develop program. It is based on blocks as well, with a program structure similar with Arduino's projects. The graphical projects developed through this technology can be translated to Arduino textual code. This feature presents the same positive aspects mentioned in *Modkit* language.

Amici does not provide a detailed graphical interactivity as the languages presented previously. Figure 3.5 shows a project developed in *Amici*'s workbench. There are two

[4]http://dimeb.informatik.uni-bremen.de/eduwear/about/

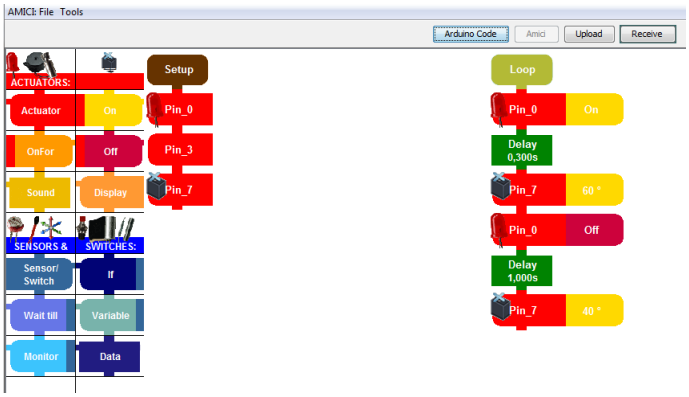


Figure 3.5: Amici example program

block's sequences, *Setup* and *Loop*. These two functions characterize the Arduino code – *Setup*'s sequence is executed only once when the program starts and it initializes variables and pin modes; *Loop*'s sequence allows the program to change and respond, consecutively. Arduino textual code is also split in these two functions.

Icons' shapes suggest which block connects to another, in a simpler form comparing these characteristic with previous languages. *Amici* focuses on robotic aspects, presenting blocks with this kind of information, such as LEDs actuators, motors. The input parameters are adapted according to block's characteristics, i.e. is defined a LED block, its input parameters could be *ON* or *OFF*. Figure 3.5 shows a LED pin which contains both status (ON and OFF); *pin 7*, representing a robot's Servo, takes degrees values as input parameters.

3.3.4 ArduBlock

Ardublock[5] is another block programming tool for Arduino. It is a plugin for the Arduino IDE, that generates code for Arduino Workbench.

Colorful blocks represent the different graphic programming elements and concepts, such as yellow blocks imply control structures (loop or conditions). Each block has a meaningful shape that defines its connection to others. Complex programs are difficult to implement, since the blocks' size limits the designing area. There are specific blocks for robotic components, such as Servos and Buzzers.

Figure 3.6 depicts an *Ardublock* program. It shows specific robotic blocks, counting with Servo and Buzzer components. It shows delay function, an Arduino concept and main feature.

Ardublock features the same goals as the previous Visual Languages.

[5] sourceforge.net/projects/ardublock/

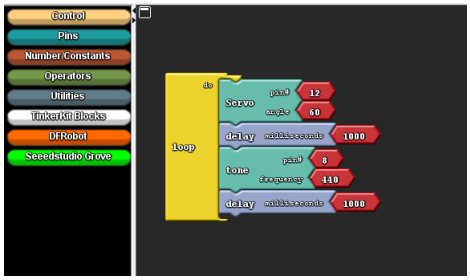


Figure 3.6: Ardublock example program

3.3.5 Minibloq

This technology[6] presents an interface different from all the others, while maintaining the use of blocks for developing programs. The blocks are represented via icons with no textual reference explaining its meaning or purpose. These icons suggest the function performed by the respective block. The pallet contains all language's blocks, describing each block function and its respective details of use, when the mouse pointer passes over a single element.

All blocks have the same shape (rectangles), and are distinguished by its own color and icon. *Minibloq* has robotic specific blocks, such as Motors, Buzzers and Servo. The remaining blocks count with general characteristics from Arduino, like time, pin voltage control (digital or analog), loops, and conditions. Figure 3.7 depicts a program containing pin voltage blocks, which change a LED state every 500 milliseconds (classic blinking behavior).

While developing a solution, it is possible to choose which Arduino board the program is going to operate. A representative figure contains the board model, indicates pins numbers, actuators and sensors connections.

As the previous languages, this tool also presents the actual Arduino code built through the developed graphical project. This tool presents a new feature – it is possible to comment blocks in the project diagram, removing them from the execution model. This characteristic allows a better control of the developed program, for analyzing and reading purposes.

3.4 Software Languages for Robots

This section analyses a group of applications for robots, that use visual elements and some functionalities to specify their programs. The study was conducted over a series of technologies, shown in table 3.1. Both children and adults are the target users for these applications, having differences in their programming knowledge and technical background. Applications' goal may vary slightly between them.

Table 3.3 shows Visual Programming Languages for Arduino. These aim to teach

[6] http://blog.minibloq.org/

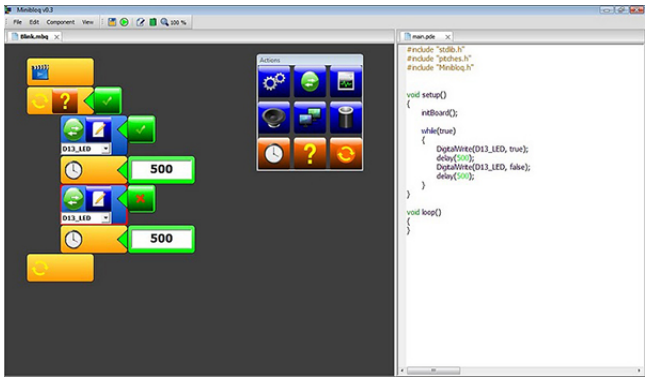


Figure 3.7: Minibloq example program

children and adults to program, in a graphical way. Some of them, show Arduino textual code while programming with visual elements, lowering the abstraction level, encouraging the learning of low level code. This characteristic violates DSL's principles. Code details should remain hidden to its target users.

Beyond Arduino world, there are robotic educational technologies with different complexity levels. Some of them are shown in table 3.2. They have various characteristics such as simulation environment (RoboMind), used for testing behaviors. Others, combine hypothetic simulation scenarios with robots actuating in the real word environment, such as Flowbotics Studio or Microsoft Robotics Developer Studio. These languages/-tools use robot-domain concepts and terms so domain experts and other users can develop new programs in a simpler form.

Following this line of thought, it is necessary to identify such language's functionalities and concepts. Each language has its own purpose, justifying its characteristics and features. Relevant details are shown in table 3.4 and 3.5. These details include the following functionalities:

- Code upload - Language Tool must provide a way to upload code to the target robot, since the robot should execute the compiled code.
- Simulation - Some tools have a simulation environment to test the developed code/-diagram.
- Descriptive Editor - The language tool should describe each one of the concepts or provide a documentation manual, so users could understand how to build a new program.
- Calibration system - Robots are composed by several components. Some of those components require maintenance, such as wheels direction. The language tool should offer this functionality via a graphical interface or some specific command.
- Server/PC control - Some tools allow controlling the robot, when it is attached to a machine, called the server.

Table 3.1: Robot Interaction Languages

Name	Abbreviation	Description
Logo[a]	LoG	Textual language designed for children, that handles a turtle-robot.
RoboMind [b]	RM	This programming language handles a robot in a virtual scenario.
leJOSNXj[c]	leJ	Java programming language to Lego MindStorms robot.
RobotLab Labview[d]	RLab	Famous Lego MindStorms' visual programming language. Behaviors are programmed through flow diagrams.
ACTOR LAB[e]	AcL	Visual Programming Language based on graphs, that determines events (Lego sensors information) logic flow.
Dialog OS[f]	DiOS	Graph visual language, that combines voice recognition with robot behaviors.
Flowbotics Studio[g]	FbSt	Time line containing robot programmed behaviors.
Microsoft Robotics Developer Studio[h]	MS	Visual Language based on graphs which allows programming robot behaviors. It is possible to display programs in a simulation environment
Scratch for Arduino	S4A	Arduino Programming Language based on Scratch Language
ModKit	MK	Visual programming language for Arduino based on blocks
Amici	Amc	Programming language for Arduino prepared with robotic concepts
Ardublock	ArBl	Plug-in for Arduino IDE, containing a graphical interface to develop Arduino programs
Minibloq	MQ	Visual Programming Language based on blocks with specific robotic icons

[a] http://logo.codeplex.com/
[b] http://www.robomind.net/
[c] http://lejos.sourceforge.net/
[d] http://education.lego.com/
[e] http://actor-lab.open.ac.uk/
[f] http://www.clt-st.de/en/produkte-losungen/dialogos/
[g] http://www.dsprobotics.com/
[h] http://www.microsoft.com/robotics/

Table 3.2: Robot Languages characteristics

	LoG	RM	leJ	RLab	AcL	DiOS	FbSt	MS
Paradigm	Textual	Textual	Textual	Visual	Visual	Visual	Visual	Visual
High abstraction level	-	-	-	✓	✓	✓	✓	✓
Periodic events	-	-	✓	✓	✓	-	✓	✓
Config -urable blocks	-	-	-	✓	✓	✓	-	✓
Containers	-	-	-	✓	-	-	-	✓
Visual metaphor	-	-	-	Dataflow and Work-flow	Data flow	Data flow	Time-line Ac-tions Work-flow	Data flow

Table 3.3: Robot Languages characteristics

	S4A	MK	Amc	ArBl	MQ
Paradigm	Visual	Visual	Visual	Visual	Visual
High abstraction level	-	-	-	-	-
Periodic events	-	-	-	-	-
Configurable blocks	-	-	-	✓	✓
Containers	✓	✓	✓	✓	✓
Visual Metaphor	Tiles-based	Tiles-based	Tiles-based	Tiles-based	Icons and tiles based

Table 3.4: Robot Language Tool features

	RM	leJ	RLab	AcL	DiOS	FbSt	MS
Code up-load	-	-	✓	✓	-	✓	✓
Simulation	Merely Simula-tion	-	✓	Real-Time control	-	Time line	✓
Descriptive editor	✓	✓	✓	✓	✓	✓	✓
Calibration system	-	-	✓	-	-	✓	-
Server/PC control	-	✓	✓	-	-	✓	✓

Table 3.5: Robot Languages Tool features

	S4A	MK	Amc	ArBl	MQ
Code upload	-	✓	✓	✓	✓
Simulation	-	-	-	-	-
Descriptive editor	✓	✓	✓	✓	✓
Hardware data	-	-	-	-	-
Calibration system	-	-	-	-	-
Server/PC control	✓	-	-	-	-

3.5 Summary

In this chapter we presented technologies suitable to children, particularly programming languages regarding the robotic domain in order to explore their visual paradigms, specific functions and features for each tool environment. As presented earlier, the languages developed to children still use many programming concepts, leaving aside domain specific notions that could help the target user to get a better acquaintance with the language and its platform environment.

Most of the studied languages exhibit an imperative programming paradigm with the intention to introduce programming concepts common in textual languages – most of languages presented above, contain some concepts/instructions with a low-level of abstraction such as loops, conditionals structures (if and else), variables and data types. Although most of these concepts have their own visual expression, the DSL requires domain specific concepts as opposed to the previously mentioned concepts. Conversely, the languages with this low-level of abstraction are able to deal with different types of robots architectures and hardware components, since they present more generic concepts in the robotic domain.

4

Domain Analysis

In this chapter we present an analysis of the target user profile and study case which will be used to evaluate our proposed solution.

4.1 User Profile

This chapter studies relevant and general end-users characteristics that will help proposing a suitable solution. It provides a brief introduction to children's aspects and technologies. There is a need to assess what is currently understood about the target users so it could be possible to create user profiles. A User Profile is a detailed description of the target users attributes that typically reflect a range, not a single attribute (e.g., ages 8 – 12). The user profile will ensure who are the target users and will help to recruit the right test subjects for future usability activities. It is vital to recruit the right users, otherwise the collected results data would be worthless [31].

Children are the main target users for the developed DSL. Therefore, it is necessary to define what a child is. A child is frequently defined by the United Nations Convention on the Rights of the Child (UNCRC) as "every human being below the age of 18 years unless under the law applicable to the child, majority is attained earlier"[32]. Age seems a limiting factor when discussing children characteristics; however, age-related definitions can not express children's gentle shifts between the states of adulthood and childhood[33]. Postmodernist movement says that childhood is a relative concept that changes "according to historical time, geographical environment, local culture, and socioeconomic conditions" [34]. Nevertheless, the best way to learn about children is to interact and spend time with them [33].

Children have their own likes, dislikes, curiosities, and needs[35]. They have similarities and differences that are studied through Child development theories. One major theoretical perspective interesting for this dissertation subject relies on Cognitive-Developmental of children.

Cognitive-Development

Jean Piaget believed that children act as scientists in order to discover how the world and environment works [33]. The *Piagetian* stages of cognitive development contain basic notions that define children behaviors, and should be very helpful in describing key stages of intellectual and language development. Table 4.1 depicts the Piagetian stages, which Piaget believed that people move through these stages of development, allowing them to think in new, more complex ways.

Stage	Age	Important aspects for Interactive Products Design
Sensorimotor	Birth - 2	
Preconceptual Thought	2 - 4	
Intuitive Thought	4 - 7	Children start to use words and symbols to express themselves. They are able to distinguish reality from fantasy as well.
Concrete operational	7 - 11	During this stage, children are able to classify things and understand notions as Conservation and Reversibility. They are still not able to think abstractly; Conversely, children develop their logical thinking at this stage.
Formal operations	11+	Children are able to think hypothetically, dealing with events and possible situations regarding the physical world.

Table 4.1: Piagetian Stages of Development

There are two particular stages interesting to analyze in this dissertation context – Concrete Operations and Formal Operations stages. The first expresses that children between seven to eleven years old, attain notions as Reversibility and Conservation – they are able to reason and think logically. However, they are not able to think abstractly and deal with hypothetical situations. Piaget believed that children only reached this new thinking way at the Formal Operation stage, beginning at age 11. This new stage marks the movement from the capability to think and reason from concrete visible events to the capacity to think abstractly, using abstract concepts to solve hypothetical problems.

Sociocultural

Urie Bronfenbrenner claim that the development of a child is made through a complex system that is divided in other sub-systems with many interactions. This sub-systems contain the relationships with individual's contexts, such as cultural, social and their immediate environment. Bronfenbrenner also affirms that a human being's environment (from the family to economic contexts) have a strong position in their life course from childhood through adulthood. Thus, it is important to study children's specific related context, such as their culture, socioeconomic status and many other aspects that are part of the human being environment.

Users characteristics

In order to design a suitable software solution for children, it is relevant to study children's characteristics such as: Dexterity, Speech, Reading, Background knowledge and Interaction style [36]. Since children's fine motor control is not equal to that of adults, input devices could lower their performance while working on software related tasks. Children's speech is not a relevant characteristic to analyze since it is merely interesting for voice recognition applications. Reading skills, being the primary way to communicate with a computer, need to be analyzed as well. That is why we should consider studying their background knowledge before any implementation effort [36]. Children often have success in learning interfaces based on familiar aspects (background characteristics) [36].

These aspects give us the power to acquire knowledge more effectively who are the target users along with their characteristics. Table 4.2 depicts the relevant issues to acknowledge before the evaluation with the users, taking into consideration children's characteristics.

Table 4.2: Children characteristics

Category	Characteristic
Physical	Age
	Gender
	Energy level
Cognitive development	Development stage
Sociocultural and Economic level	Socioeconomic Status
	Relationships
Background Knowledge	Academic year
	Subjects' grades
Technological Background	Computer Access
	Internet Access
	Other technological devices

If children have more practice and experience with computers and technologies, they tend to be more successful in new computer-related tasks [37]. The technological background give us the information about children's interactivity with technical devices which

may affect (positively or negatively) their cognitive development [38]. Furthermore, it is understood that children are more likely to work and play in groups, in a single computer [36].

For evaluation studies, however, it is often necessary to be able to make some assumptions about a cohort of children with respect to their abilities and skills at a given point in time [33]. During these studies, children might become nervous at the thought they are being tested. Each child has a unique temperament that could have a significant effect on both evaluation cases and the consequent results. There is a need to reduce the effects of temperament on evaluation studies.

The robot model in children mind

Since this dissertation assesses the children's conceptions and their ability to create artificial robot behaviors, it is important to examine how this type of activities alters the children's mind and their mental development. According to study [39], 5 to 7 years old children lack knowledge in relation to the robot's ability to perform actions and behave autonomously. The adaptivity behavior (hypothetical situations like collision avoidance) that robots should run, is a difficult task to understand by children, according to the mentioned study and its results. Although it is difficult to teach this type of behaviors to young children, they showed successful performance in rules construction and thinking, as the use of technological languages. Once again taking this study into consideration, we may conclude that suitable tools and tasks allow children to accomplish, construct and reflect their understanding on behaving agents/objects.

4.2 Case study

This dissertation proposes the development of an educational and entertainment DSL for children, regarding the Robotic domain. The solution's goal is to provide an effective, efficient and an user-friendliness way for children to specify robot behaviors. It is necessary to choose a target platform, so it could be possible to develop and evaluate the proposed DSL. Therefore, a company called *Artica* has provided a Robot with a set of specific components. The robot, named *Farrusco*, is built on an Arduino board with a predefined configuration of its modules. It encompasses several actuators motors and sensors. *Farrusco* is shown in figure 4.1. It contains the following components:

- One infrared distance sensor is able to detect the presence of nearby objects without any physical contact;
- Two bumpers detect robot's collisions, through physical contact;
- Two direct-current motors for each wheel that controls *Farrusco* velocity and direction;
- One Led attached to the Arduino board;

- One Servo supports the infrared distance sensor, setting a position at a particular range.

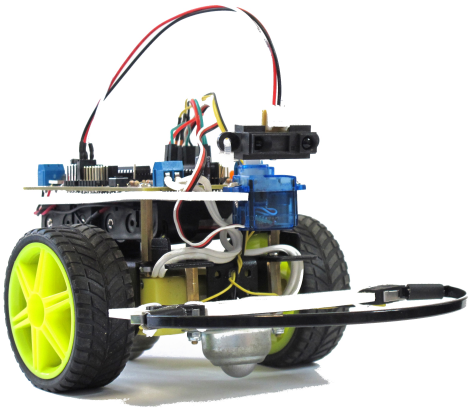


Figure 4.1: *Farrusco* robot

This is the basic *Farrusco* configuration and each sensor or actuator has its own pin at the Arduino main board.

4.2.1 Arduino Platform

Arduino[1] is an open-source electronics prototyping platform, based on Atmel Atmega microcontrollers[2]. It is intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments, combining easy-to-use hardware and software. It has its own programming language and *bootloader*, ready to receive and execute previously-compiled programs.

Before Arduino, developing simple projects involving computer science programs with electronic hardware platforms was difficult to achieve. Microcontrollers were intended for domain experts, since its manufacturer's manual was full of specific terms and extensive domain-data information. These components only were used in the industrial sector by expertise teams. Thus, microcontrollers were far from people who have no experience in this specific domain. *Arduino* has revolutionized microcontrollers' development, bringing the power of building electronic devices to the common citizen.

The industry and its market are still led by CPLD's [3]. CPLDs are robust, reliable and flexible devices in the industrial domain, where these factors are extremely important. Thus, CPLDs' high cost corresponds to the quality of service they provide. These components require an extensive developers' training, as they offer several complexities and architectural differences that vary depending on the manufacturer.

Arduino's purpose is not to replace this kind of microcontrollers. Students as the target users, should be able to develop electronic projects in a simpler way. This is an

[1] http://arduino.cc/
[2] playground.arduino.cc/Main/AVR
[3] Complex programmable logic device

important feature of Arduino comparing it with the original CPLDs process development. Another core strength of Arduino is the community of users who develop Arduino projects, contributing ideas, thoughts and code to the Arduino Project.

The Arduino integrated development enviroment (IDE) is derived from the IDE for the Processing programming language. It comes with a software library named *Wiring* which makes common input/output pins' operations much easier.Arduino programs are writen in C or C++ and users need to define two main functions to make a runnable cyclic executive program:

- setup function: it runs once at the start of a program that can initialize settings;
- loop function: it is called repeatedly until the board powers off.

Sensors and actuators are attached to the Arduino board through its available pins. After establishing the physical connections, there is a need to define those in the program code. A sensor inputs information to the board, so the respective pin is configured as an *INPUT* pin. Actuators produce environment changes or physical behaviors, such as motors or lights; these need to be configured as *OUTPUT* pins, since they perform actions. These configurations are made in the *Setup* function presented previously. Arduino contains PWM pins. PWN, or Pulse Width Modulation, is a technique for getting analog results with digital means. The input voltage can be modulated if an actuator is attached to a PWM pin. This is used for controlling a Led's density light, or the motors' speed.

The IDE is also capable of compiling and uploading programs to the Arduino board.

Arduino board can integrate new circuits, plugged into the supplied Arduino pinheaders. These circuits are named *Shields*, and they can provide motor controls, radio and GPS signals, LCD displays and ethernet. With this characteristic, it is possible to create easily a heterogeneous device made by several different components.

4.3 Domain Model

The domain analysis was conducted with the domain experts collaboration, so it could be possible to identify concepts, terms, notions and ideas present in this work area. The robot itself, *Farrusco*, contains several components and features to consider. The DSL should express common robot behaviors and it must be prepared to a set of robotic components. The *Feature Model* 4.2 depicts required characteristics which the DSL must provide.

The detailed analysis of Robotic technologies and Programming Tools for children was specially useful. Designing the DSL's basics concepts became easier to deploy, once the domain study was completed. We took in consideration the Robotic technologies studied before as the programming tools mentioned in the previous chapter.

The domain experts were used to program robot's actions through behavior tree structures via textual code. They claim that behavior trees are a powerful way of organizing a collection of states, making this structure more suitable for representing complex

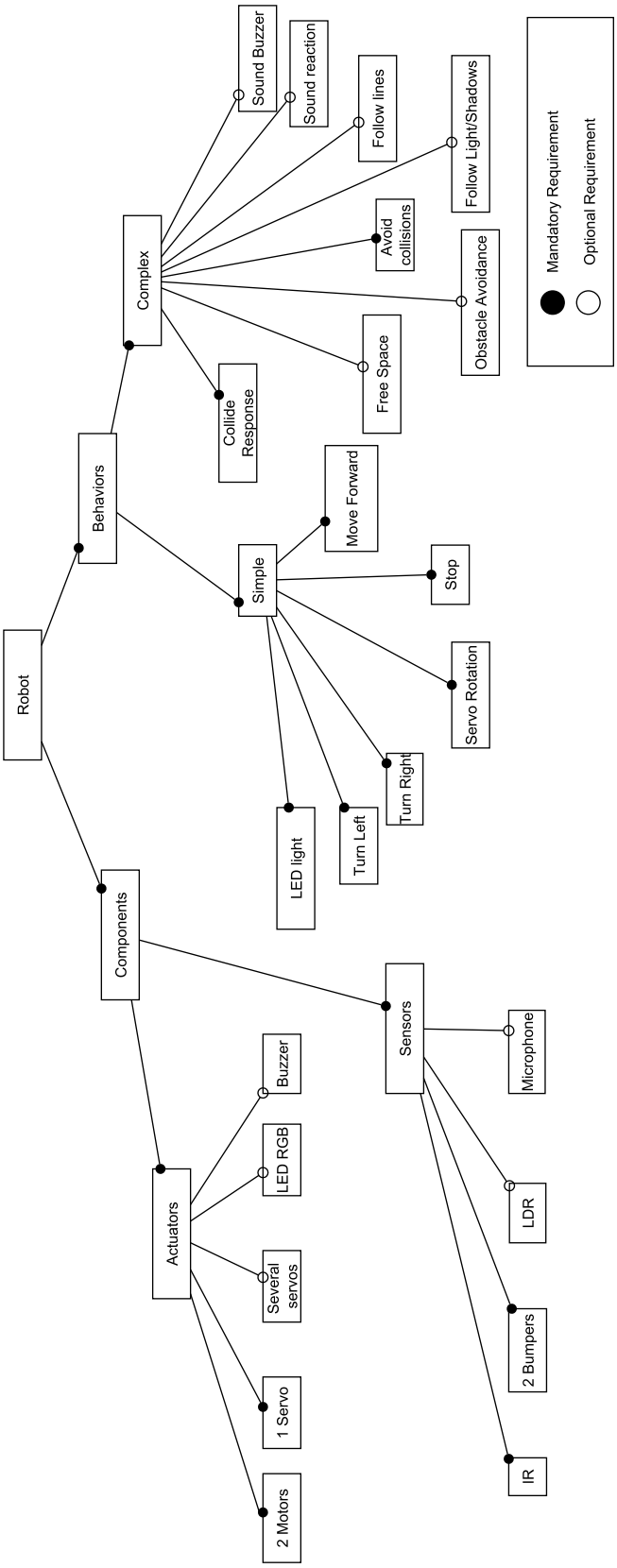


Figure 4.2: Feature Model

and potentially parallel robot behaviors. Their advice made a big decision step in the DSL process development – behavior trees has been chosen as the visual programming paradigm.

4.3.1 Robotic Terms and Concepts

Robotics is a wide domain containing several specific terms, since it deals with design, construction, operation and application of robots. This section describes the most used concepts used in the robotic area, and specifically the presented domain case study robot, *Farrusco*. The following set of terms contains components and common robot behaviors definitions:

- Bumper, is a collision sensor coupled in the front side of robot. It works like a mouse button, with a mechanical switch that usually corresponds to a collision situation. When pressed the robot should behave as it is supposed to;
- Infrared distance sensor, is an electronic sensor that measures infrared (IR) light radiating from objects in its field of view. It should avoid collision with an obstacle;
- Servo provides position control, in this specific case, the infrared distance sensor position. It uses an electrical motor as the primary means of creating mechanical force;
- DC Motor, or Direct-current motor, is a mechanically commutated electric motor powered from direct current. Common robots have two DC motors, one for each wheel. They provide direction and speed control;
- LDR or light dependent resistor is a sensor which measures the incident light intensity from the environment;
- Collision detection is a predefined behavior that should be executed whenever the robot collides with an obstacle. It should move backwards and turn to a different direction;
- Follow Light is a behavior that autonomously controls robot direction and speed, to follow a light-emitting device. This type of behavior is also called *phototaxis* — movement toward a light source. The robot should contain at least one light sensor;
- Follow wall is yet another robot behavior, that controls robot's motors speed and direction. The robot should follow a wall without colliding.
- Free Space is common behavior implemented for robots. It combines the distance sensors information with motor speed and direction. The robot should move to a wider area without obstacles. This takes place whenever the robot *senses* an obstacle;

- Obstacle Avoidance is the task of satisfying a control objective. It is often achieved by direct sensing of the environment through distance and obstacle sensors. The robot should round the obstacles, instead of moving to a wider area.

4.3.2 Domain Model Specification

Domain Model formalizes the characteristics related to this specific robotic domain. It counts with the behaviors and components that encompass the robot itself and its own purpose to the world. The domain model provides an abstract structural view of the domain.

The developed domain model is depicted in figure 4.3.

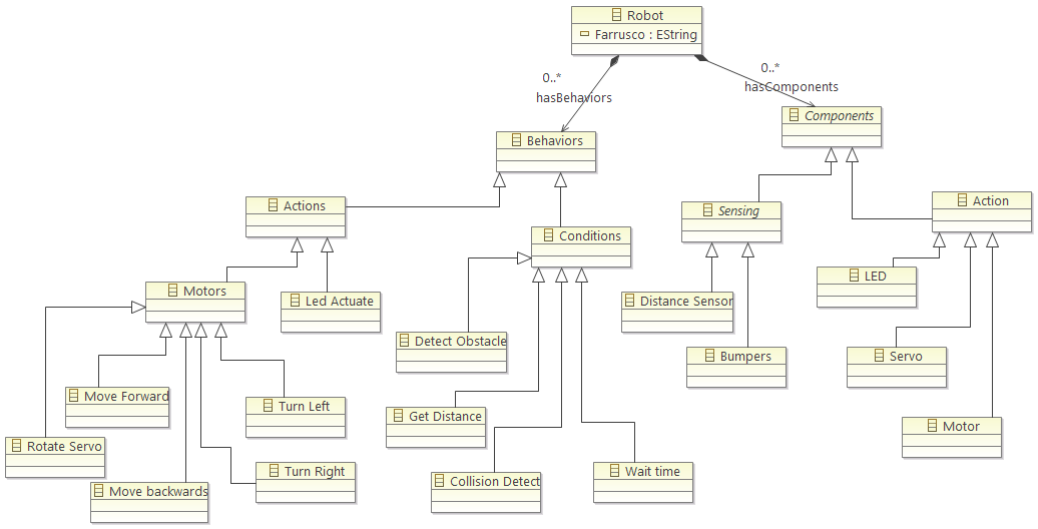


Figure 4.3: Domain Model

5

Language Design

This chapter presents the DSL design. This process takes advantage of the domain analysis taken from section 4.3. The involved topics from that section can be used to design the DSL inner structure's models and concepts. Considering these arrangements, the following section contains the metamodel itself and the ideas it contains.

5.1 Metamodel

Material studied from domain analysis conveyed concerns that affect the construction of the DSL. The metamodel represents the join between the domain analysis, in this case the target users as children and their capabilities, and the way behavior trees are used to represent robot behaviors. The Metamodel describes the DSL and the abstractions of its target platform, the Farrusco itself. Since this is the base of the whole process of deploying the language, the metamodel should be well designed, because one slight modification in the metamodel concepts can bring severe future changes.

As explained in section 2.5, behavior trees have two types of nodes – internal nodes and leaf nodes – that define a behavior, independently of the operating background. In our case, the proposed behavior tree structure controls the components from Farrusco Robot. Those are extremely important concepts because they compose the inner organization of the language.

The metamodel encompasses Farrusco concepts embedded in the behavior tree paradigm and its abstract syntax is described in the subsequent section.

5.1.1 Abstract Syntax – Relations and Properties

Farrusco, as the main class of the metamodel, aggregates three concepts of the behavior tree paradigm: nodes, siblings, and children. A behavior/instance of the model may contain several nodes, connected as either siblings or children. The Sibling class defines a node that has another node as its sibling. The Children class determines the correspondent children nodes for a specific node. These rules and properties that define the node concept are depicted in the metamodel fragment in figure 5.1.

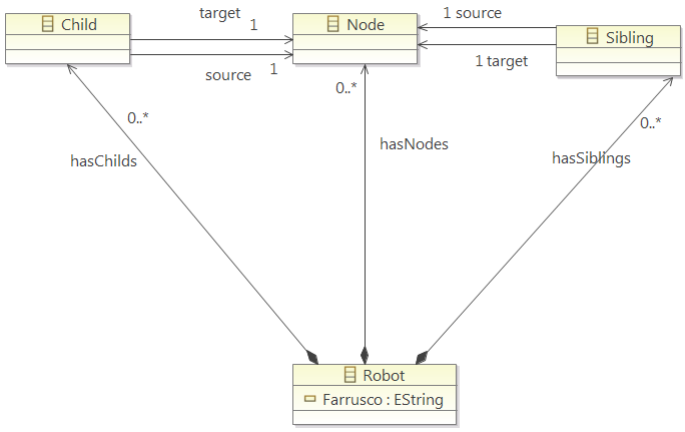


Figure 5.1: Node and links Metamodel fragment

There is the need to distinguish the different types of Nodes that compose the behavior tree, and the language itself. The Node class is a generalization from two classes that correspond to specific blocks of the diagram – the Control and Leaf classes. Control nodes compose and delineate the execution of behavior trees. Leaf nodes correspond to actions or sensing behaviors, because these are the domain target nodes.

The metamodel fragment from figure 5.2 corresponds to the Control concept and its generalization. Priority Selector, Parallel and Sequence are the three internal nodes that control the execution of the behavior tree. These are the only nodes that can have other nodes as children – leaf and control nodes. Control nodes' details and specification are explained in the Mapping Syntax section.

Figure 5.3 depicts action and sensing nodes, i. e. the leaf nodes for the behavior tree. Those represent the components that set up Farrusco. That is the reason why there is a class named Components, that is the generalization from Sensing and Actuate classes. Sensing manages the sensors from the robot, and Actuate deals with Farrusco's actuators. Leaf nodes are directly related with Farrusco's modules shown in Domain Analysis Chapter.

Given that leaf nodes have configurable factors, it is important to specify data types that should help the user handling the right parameters for a specific node. An enumerated type is a data type consisting of a set of named values. These named values are

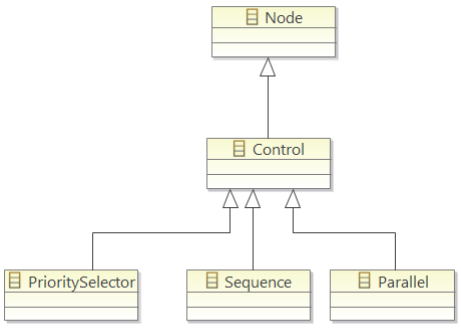


Figure 5.2: Control Nodes Metamodel fragment

identifiers that represent domain specific concepts in order to achieve a greater abstraction towards the language user. For example, in the case of a LED – which can either be on or off – a boolean type could be used to represent these states, but it would be clearer for the user if instead of using this primitive type an abstraction were to be used, thus allowing the user to select from two states "On" or "Off", avoiding that he has to opt between true or false. This simple example gives us the information that node parameters have their importance in the metamodel design. Each node requires its specific enumerated type so the user could select the right parameter inside a range of choice. Considering this, four new data types belong to the metamodel, where each one is used in an attribute for a specific node:

- LedState has values "On" and "Off", and it is associated with an attribute for LED class. It determines if the LED should be turned On or turned Off.
- ChooseBumper has two values – "Right Bumper" and "Left Bumper". This enumeration was created so users could select which bumper to use in the Bumper node.
- CompareDistance has two enumerated values that define how the distance value should be compared. This condition can either be "smaller than" or "greater than", meaning that the distance value triggers an action when the aforementioned condition is met.
- DirectionType has five enumerated values that define how Farrusco should move. "Turn Left", "Turn Right", "Move Forward" and "Move Backwards" automatically set both motors to run in the corresponding direction. The fifth enumerated value named "Manual Direction" gives an option to define manually both motor power attributes.

These enumerated types are depicted in the metamodel fragment from figure 5.4.

This leads to leaf nodes' specification, beginning with the actuators components from Farrusco. The following metamodel classes share similar names and concepts with the

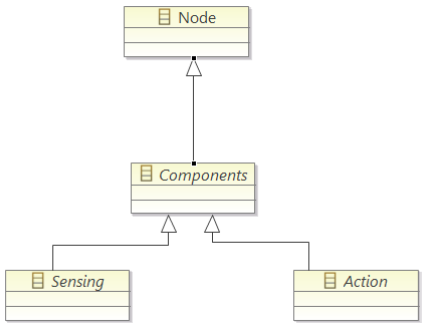


Figure 5.3: Leaf Nodes Metamodel fragment

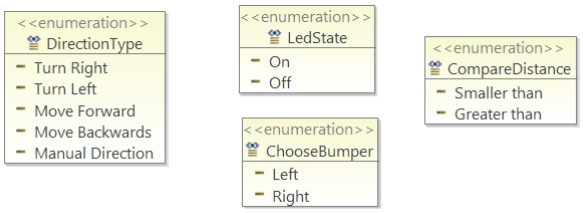


Figure 5.4: Enumerated types from the metamodel

components that constitute Farrusco. The Actuate class manages three main physical outputs from Farrusco, and it is a generalization from three classes:

- Motors class represents the Farrusco's motors. There are two motors, one for each wheel, so is important to define two attributes that control the power for each motor:
  - "Left Motor" is an integer attribute that controls the left motor power. It could vary from -255 to 255. Zero value stops the motor and negative values set the motor to rotate backwards. Positive values will set the motor to rotate forwardly.
  - "Right Motor" corresponds to Farrusco's right motor and has the same proprieties of the previous attribute.
  - "Direction" is a DirectionType (DirectionType datatype) attribute, where the user should select one option to control Farrusco Motors.
- Servo is composed by three integer attributes:
  - "Maximum Position" is an integer that defines the maximum position for Servo. It takes 180 as maximum value.
  - "Minimum Position" is an integer that defines the minimum position for Servo. It takes 0 as minimum value.
  - Pulse, an integer that manages Servo's position over time. It may vary from 0 to 180.

- LED has only an enumerated type attribute (LedState datatype), which can either be set to On or Off.

This fragment from the metamodel is illustrated in figure 5.5.

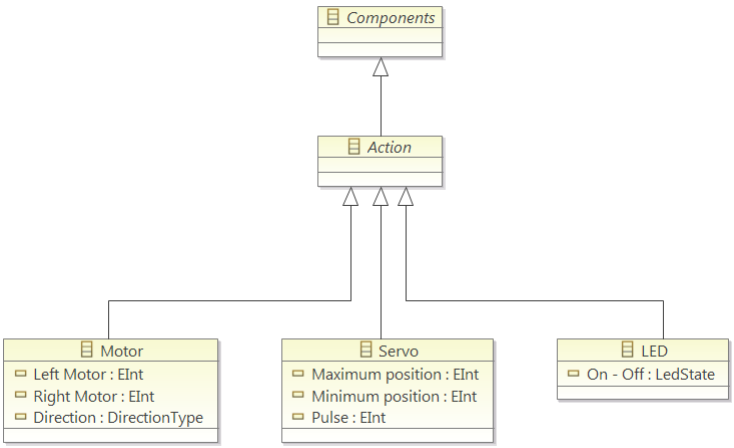


Figure 5.5: Components Nodes Metamodel fragment

Sensing class is similar to the previously introduced classes, which is once again a generalization for Farrusco's sensors. Sensing class is depicted in figure 5.6. Farrusco's architecture has two different types of sensors, each represented by its own class:

- Infrared Distance Sensor class has two attributes:
  - Distance, an integer that corresponds to the aim distance.
  - DistanceType is an attribute of "CompareDistance" enumerated type. The user may select how Farrusco should compare the measured distance with the selected from the previous attribute.
- - Bumper class has only "ChooseBumper" as attribute that defines which bumper to use – Left or Right.
- - Wait class has an integer to express the seconds the robot should wait between tasks.

Farrusco's components define most of the language's characteristics. The introduced details define a metamodel opened to new hardware components.

Considering these metamodel aspects, it is safe to say the DSL design was made through a top down approach, considering behaviors that Farrusco should perform as an objective to be achieved. Each of Farrusco's components is deemed in the designing process as sub-parts of the DSL. The completed metamodel is depicted in figure 5.7.

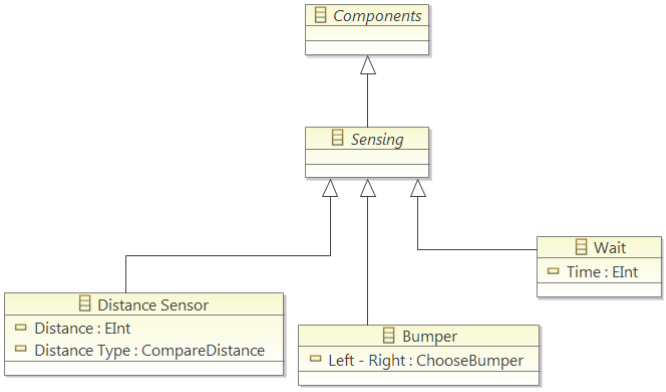


Figure 5.6: Sensing Nodes Metamodel fragment

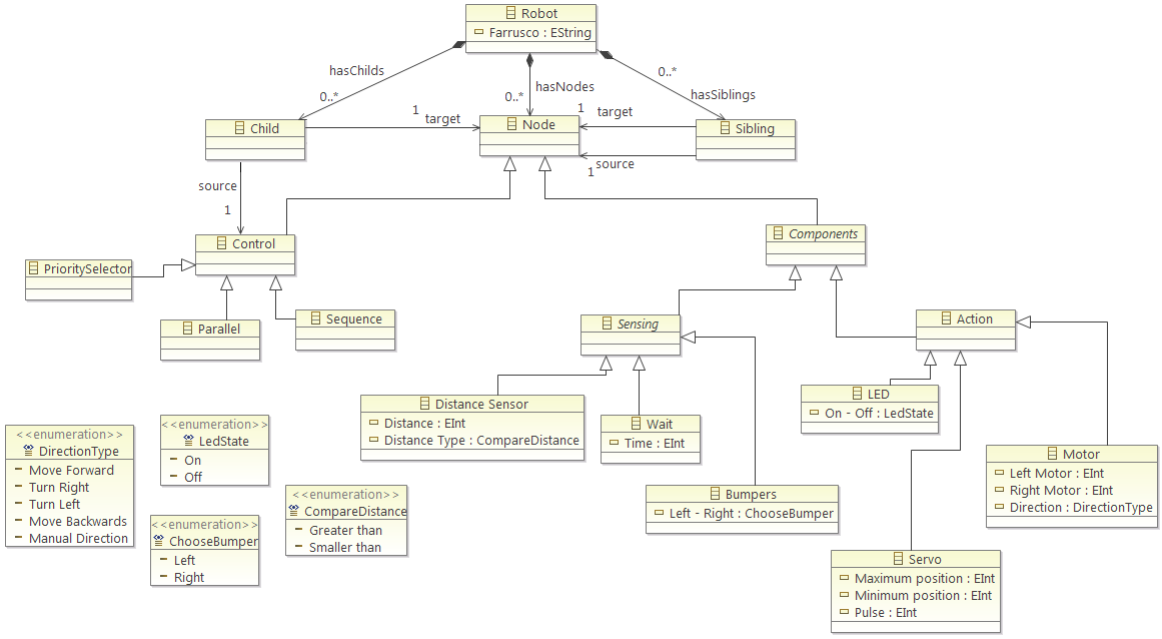


Figure 5.7: Complete Metamodel

5.2 Concrete Syntax

The previous section presented metamodel design including specific features and concepts. Those characteristics are directly related with visual representations used in the graphical language interface. This section shows the mapping between metamodel concepts with visual interface elements from the graph editor.

The aforementioned metamodel was designed in Ecore-based used by Eclipse Modeling Framework (EMF)[1]. The developed graphical language was based on the GMF[2] /

[1]http://www.eclipse.org/modeling/emf/
[2]http://www.eclipse.org/modeling/gmp/

EUGENIA[3] that allows the creation of new robot behavior models. This Ecore Metamodel was enriched with Eugenia annotations, which link to rules that describe how the metamodel entities and relations are represented in the Graphical Editor. Each metamodel class has its own Eugenia annotation, which results in a visual representation for that specific concept. Therefore, the use of nodes and links as connections is adopted as the base for the language visual design. This means that each concept shall have its own graphical representation. Eugenia annotations hold those language icons and figures as well. The chosen icons and figures that represent the inner concepts from the metamodel took in consideration the target users' characteristics such as age and the low levels of programming knowledge.

Analogously with the metamodel section, the concepts are divided in three main classes. This section will describe the Eugenia's annotations made for each Ecore element from the metamodel. The mapping made through Eugenia Tool, between Control nodes from the metamodel and visual interface is depicted in the table 5.1.

Table 5.1: Control Nodes concrete syntax

Metamodel Element	Visual Name	Icon	Description
Sequence	Sequence		Its children nodes are performed in a sequence of steps.
Parallel	At the same time		Parallel node's children are run simultaneously.
Priority Selector	Decider		Earlier children have higher priority to get run (but may fail). Later ones are less desirable but presumably more likely to work.

Table 5.2 shows the annotations for the leaf nodes.

[3]http://www.eclipse.org/epsilon/doc/eugenia/

Table 5.2: Leaf Nodes concrete syntax

Metamodel Element	Visual Name	Icon	Description
Motors	Motors		It controls Farrusco's motors and direction.
Servo	Neck		Controls Farrusco's Head.
Led	Light		Switch light power.
Wait	Wait		Farrusco waits for a specific time period.
Bumpers	Collision Sensors		Check if Farrusco collided with some obstacle.
Distance Sensor	Distance Sensor		Checks the presence of obstacles.

Language's connections – children and siblings links – are represented in Table 5.3.

Table 5.3: Links concrete syntax

Metamodel Element	Visual Name	Icon	Description
Child	Child		The link made for nodes that have the same parent.
Sibling	Brother		The link made from a parent node to a child node

6

Implementation

This chapter discusses the DSL development process and its specific details. The DSL was developed through a model-driven approach, using Graphical Modeling Framework and Eugenia Tool for Eclipse Modeling environment. These technologies encompass the whole development process of the DSL. The starting point for the implementation took place once we studied the target domain and its characteristics. As we will have the opportunity to observe in the next section, the metamodel was developed in the Eclipse workbench. The metamodel required the introduction of additional properties and annotations in some of its elements due to specific technology constraints that are necessarily different in other language workbenches. Therefore, some of the presented details in this chapter have tool dependencies, since those represent the specific implementation of the language semantics and concrete syntax. The following section shows the implementation steps and activities taken in the development process.

6.1 Development Process

Figure 6.1 depicts the development process taken for the implementation of the DSL. The metamodel was created with EMF framework for Eclipse, and respectively annotated with Eugenia tool. These annotations and constraints represent the language semantics and syntax. Eugenia allows to embed language concrete syntax and editor details directly on the metamodel – in the form of annotation for each of its elements. Once the metamodel is properly annotated, it is possible to generate the graphical modeling editor through Eugenia functionalities, which is refined with EOL expressions in order to arrange the pallet icons and elements. EOL also give us the opportunity to implement a function which applies the EGL templates to the user instance model with the purpose

to generate Arduino textual code. The behavior tree data structure was implemented during the presented process, since we needed to modify small details from it along the whole development process.

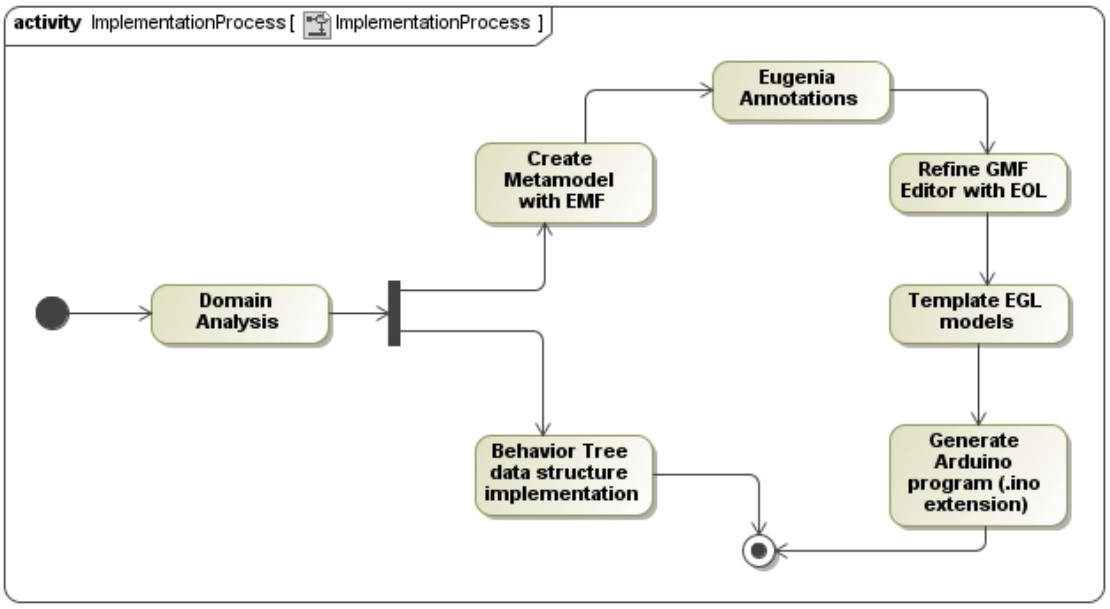


Figure 6.1: DSL Implementation Process

Figure 6.2 shows that the user expresses his mental robot behavior model through the editor – the Visualino Workbench. The target user manipulates the editor in order to create a new model, which will contain a robot behavior. Code generation templates are applied to the user instance model (the robot behavior) so it could be possible to generate Arduino textual code. The Arduino IDE will compile the generated code and upload the behavior to Arduino board through the USB connection made between the user PC and the Farrusco robot/Arduino board.

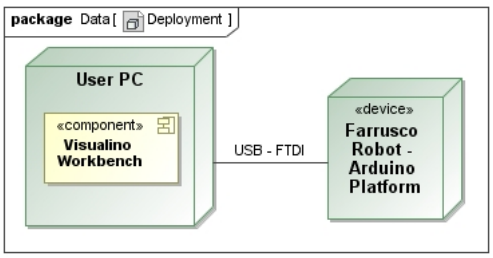


Figure 6.2: Devices involved in the DSL

6.2 Editing the metamodel with EMFATIC and Eugenia tools

Eclipse Modelling Framework was used to build the DSL along with EMFATIC text editor. The EMF file described textually the metamodel concepts, rules and proprieties. Listing 6.1 represents a leaf node (Led) of the metamodel, along with its attributes and proprieties. The first two lines describe a Eugenia annotation that represents the name, description and figure that the concept/node will use in the DSL visual editor. The following lines describe the class element with its own attribute. This attribute has a predefined value that could be adjusted by the user, when creating this specific node. In this example, the default value for the Led block is "On". If the user wants to turn off the Led, he will need to change this attribute, after creating the node in the model.

Listing 6.1: Annotation for Led leaf node

```
@gmf.node(figure="figuresPlan.figures.LEDFigure",tool.name="Light",
tool.description="activates␣deactivates␣the␣LED␣from␣Farrusco")
class LED extends Action {
  attr LedState On_or_Off = "On";
}
```

The previous example showed how a leaf node is implemented in Eugenia and EMF tools. Links are important concepts to exemplify, since they maintain the order and the connections inside a behavior tree model.

Listing 6.2: Annotation for child link

```
@gmf.link(source="source", target="target", style="dot", color="100,149,237",
 width="2",
 tool.description="This␣link␣connects␣a␣father␣node␣with␣its␣child")
class Child {
  ref Control[1] source;
  ref Node[1] target;
}
```

Listing 6.2 defines the child link concept of the metamodel. As the previous example, the first three lines describe the Eugenia annotation that represents the visual appearance of the link type (children link) in the DSL visual editor. It expresses that a link has a control node as source and a generic node as target, because the child can be either a control or a leaf node.

The DSL development started with the metamodel design, already explained in Language Design chapter. Eugenia simplifies the DSL creation process by having editor and concrete syntax details directly in the metamodel – in the form of annotations.

6.3 Code generation

This section details the code generation process from the user instance model to actual code. Arduino platform executes C++ code, so the generated code shall have the same

characteristics.

Considering behavior tree as the visual paradigm for the developed DSL, the generated code should be prepared to represent the modeled behavior. Thus, it was necessary to develop the behavior tree structure which should include the Arduino library and Farrusco's specific components and behaviors. This data structure is the solution behind the code generation process. Thus, a new Arduino library specifically for Farrusco was implemented, containing behavior tree control and leaf nodes corresponding to Farrusco behaviors. Before implementing the templates for code generation, behavior tree data structure was designed and ready to operate Farrusco behaviors in Arduino. The following section describes this data structure and the choices made along with it.

6.3.1 Behavior Tree data structure

As previously mentioned, Arduino board should execute a behavior tree object, containing a collection of states and decision processes. Therefore, an existing Behavior tree C++ library, named Libbehavior[1], was used for implementing Farrusco's components and behaviors along with StandardCplusplus[2] library for C++ standard functions and structures, such as *std::vector*. These two libraries offered an opportunity to execute and implement behavior trees on Arduino boards.

A Behavior tree makes decisions based on the current state and possible input information, e.g. sensors data. It has two types of nodes – control and leaf nodes. Leaf nodes represent domain specific actions, such as control robot wheel motors or gathering sensors information. Some of these actions could be happening simultaneously, in order to produce an interesting behavior instead of a linear sequence of actions. These nodes decide which of their children nodes to execute, using different schemes and approaches, such as parallel or sequential execution. Each nodes can return Running, Success or Failure states:

- *Running* state, prevents a node from terminating its action. It remains running until other node breaks this action;
- *Success* state returns a success message informing that everything went as expected;
- *Failure* state is used for conditions, e.g. if the robot does not detect an obstacle, the corresponding node returns Failure. This means that other node should be executed.

Some original control nodes from the library were removed so target users could understand and use control nodes' basic concepts. So, the available control nodes are:

- Sequential Node runs all of its children nodes in a sequential order;
- Parallel Node executes its children at the same time;

[1]https://code.google.com/p/libbehavior/
[2]https://github.com/maniacbug/StandardCplusplus

- Priority Node maintains an ordered sequence of its children, where earlier ones have higher-priority. This node is used for conditions, i. e., earlier children contain conditional behaviors that check if something occurred; if so, the developed behavior shall be executed, otherwise priority node executes sequentially the following children, until one of them succeeds.

Parallel execution can be challenging for Arduino boards, since the hardware is prepared for simple behaviors. This particular node can cause an overhead in Arduino system, because all of its children are called at the same time, containing other possible parallel nodes. Nevertheless there were no problems of this kind in the developed behaviors for Farrusco.

Behavior tree is executed through its first defined node. This node is called root of the behavior tree and executes its children. This process continues down the behavior tree until execution reaches leaf nodes. A leaf node will awake actions in the robot, since they are implemented with Arduino instructions specific for each component and function. These nodes take arguments in the developed leaf nodes are:

- Motors leaf node returns *Success* state when the chosen speed is applied;
- Servo node activates the motor from Farrusco's neck returning *Running* state so it continues to move in that specific movement;
- Led node returns *Success* when the user action (On or Off) is performed on the robot;
- Wait node returns *Running* state until it reaches the value specified by the user. When this occurs, it returns *Success*;
- Distance sensor node returns *Success* if it comprehends the parameters defined by the user. Otherwise (in case the robot does not detect any obstacle) returns *Failure* state;
- Bumper node returns *Failure* states if the robot does not collide with any obstacle. Otherwise, it returns Success.

6.3.2 Generating code

The generation process is executed by the end-user, so he can upload the designed behavior to Arduino board. The user develops an instance model through the language editor, which is represented by an *XMI* textual file. This file is a textual representation of the visual model created by the user. It contains all the necessary information from the model, such as elements, relationships and attributes. It is assumed the user created a well-formed tree, with no isolated nodes. An *EGL*[3] implemented template was used to generate code through the user's model and the original language metamodel. This

[3]http://www.eclipse.org/epsilon/doc/egl/

particular template retrieves the element nodes from the *XMI* file that expresses textually user's instance model.

XMI file is composed by three types of elements corresponding to those implemented in the metamodel:

- Node element defines a specific node (Control or Leaf node) which may contain user parameters in the form of attributes;
- Child represents a link between two nodes, containing specific parent and child node identification;
- Sibling expresses the order between two nodes. Once again, it contains identification from both nodes (source and the target Id's).

The *EGL* code generator file begins with a static section which contains text that will appear verbatim in the final output. This static section defines libraries and main functions (*setup* and *loop*) of an Arduino program. It follows with a dynamic section which encompasses *Epsilon Object Statements* in order to access instance model elements. This section retrieves the behavior tree nodes present in the user's model, declaring the behavior tree objects one at a time. It is important to take caution when declaring a leaf node object, since those kind of nodes commonly have attributes/parameters modified by the user. Listing 6.3 declares the nodes defined in the user instance model. *DeclareNode* function is implemented in listing 6.4 . Taking the *Priority Node* as an example, listing 6.5 shows the function that prints out that node to the Arduino file. These functions merely illustrate how EGL works in Visualino, in purpose to generate an Arduino File which corresponds to the target user instance model.

Listing 6.3: Declare Nodes iteration

```
[%for(node in all_nodes){%]
  out.println(node.declareNode());}
[%}%]
```

Listing 6.4: Check which node to declare

```
operation Node declareNode(): String{
  switch(self.type.name){
    case "Motor" : return self.declareMotors();break;
    case "LED" : return self.declareLED();break;
    case "Bumpers" : return self.declareBumper();break;
    case "Distancia" : return self.declareIRSense();break;
    case "Espera" : return self.declareWait();break;
    case "Servo" : return self.declareServoRange();break;
    case "Prioridade": return self.declarePriority();break;
    case "Paralelo": return self.declareParallel();break;
    case "Sequencial": return self.declareSequential();break;
  }

}
```

Listing 6.5: Example to declare Priority Node

```
operation Node declarePriority(): String{
   return "BehaviorTree::PriorityNode*␣pr"+self.id.substring(9)+
        "␣=␣new␣BehaviorTree::PriorityNode();";
}
```

Since a behavior tree encloses a specific order and structure, it is necessary to find the root node of the model. Typically, the root node is the first node element defined in the *XMI* file. It has one specific characteristic – it is the only one that has no parent node. This is achieved by iterating over *node* elements and for each one of them, check if it has a parent node. Once the root node is discovered, it is possible to declare the behavior tree structure.

A depth-first search algorithm recursive-function traverses the root's children nodes through Child and Sibling elements from *XMI* file. Athwart this mechanism it is possible to generate and maintain the behavior tree data structure in an organized/logical way.

Once the function fills the data structure according to the diagram designed by the user, it is essential to execute this object in the loop function of Arduino (Farrusco will keep executing this generated behavior until the user plugs it off). This is accomplished through another static section that calls the behavior tree object inside the Arduino's loop function.

The generated text is exported to an Arduino file (*.ino* extension).

6.4 Editor

As previously said, the editor workbench was created using Eugenia Tools functionalities. It has a similar interface with Eclipse environment, containing a design surface area, a pallet (or toolbox), and a properties window. This can be observed in figure 6.3.

Additional customizations were made with EOL in order to increase user usability – the palette was partitioned in four sections. Since behavior trees have two types of nodes, control and action sections were created. Control nodes belong to the control section of the palette; Leaf nodes belong to the action section; Children and Sibling links have their own section as well.

Since the editor is the main communication tool between the user and Farrusco, its interface elements (control and leaf nodes) should be named as familiar user concepts, e.g. Servo, the motor which controls the distance sensor, is expressed as *Neck*.

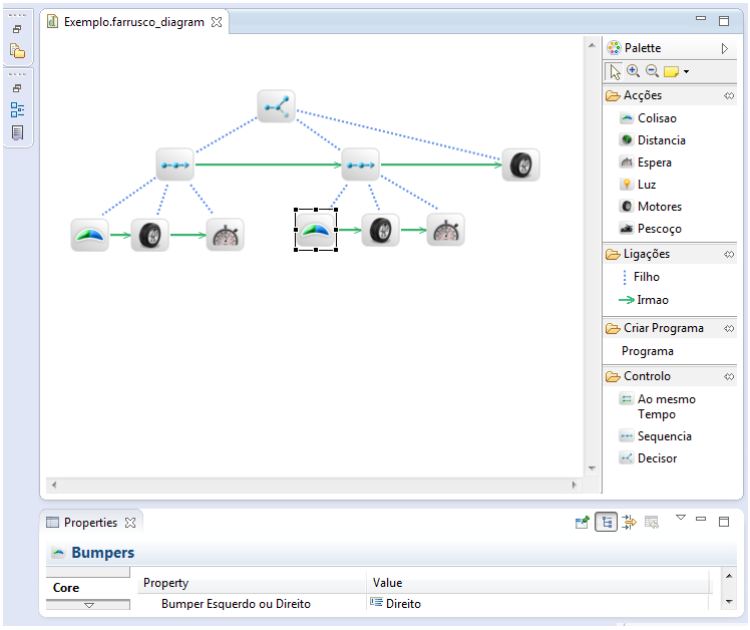


Figure 6.3: Editor Interface

The properties window displays configuration parameters of the current selected node. Generating code option is also offered by the palette. This customization accesses the instance model developed by the user, applying the code generator template. A new file is generated and shown in the project explorer view, that should be compiled and uploaded by the Arduino IDE.

7

Evaluation Process

Without an evaluation in the overall DSL development process we are still incomplete. It is necessary to prove the produced DSL has managed to overcome its requirements and the goals previously defined. Therefore, we need to test it with the target users. It is interesting to analyze and study the projected DSL with a particular age group of children. This group must be included in the two last Piaget stages previously mentioned – Concrete and Formal Operational Stages (7 - 11 years old). This is why our evaluation includes children from eight years old onwards, which give us the power to observe if there is an increasing productivity when children of different ages and maturity levels interact and develop robot behaviors with this DSL. The main objective is to evaluate the DSL usability, along with its standards. Taking this in consideration, we need to specify once again the meaning of usability. ISO 9241-11 says that usability is "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". The number of errors will measure effectiveness. On the other hand, efficiency can be and will be measured by the time elapsed during a task. Satisfaction can be measured by asking the participant if he had problems with the language after the test. Accessibility is still an important characteristic that focus on learnability and memorability of the language terms. ISO 9126 extends this definition with the notion of Goal Quality. The user in its real context of use can evaluate this perception through an actual utilization of the product. Therefore, the evaluation is made through study cases, where answers regarding Visualino usability can be found. The process needs to register these flaws by taking notes or perceive errors or problems from video records, taken during the experiment. These cases need to be well prepared since the participants are children, and they require a special attention. This special care for children is justified at the profiling section (4.1).

7.1 How to evaluate

In the previous section, it was not mentioned how to perform reliable study cases that could give us results and knowledge on Visualino usability. Real life test cases can help with this problem. Therefore, classes and exercises were prepared for children with one goal – evaluate Visualino usability. One way to achieve this objective was to compare Visualino with other robot specific languages for children, in the same conditions. Those conditions include the domain applied in Visualino development: a language that involves robots and is adequate for children. One famous robot programming language is called RoboLab language enclosed in the Lego MindStorms NXT Software. This tool/language has similar features with Visualino, such as the Lego MindStorms robot with Farrusco. It seems pertinent to compare Visualino with Lego MindStorms since both of them have many goals in common.

Lego MindStorms language is not enough to compare Visualino usability, so an additional third language was included in the research. This language, named ArduinoFlow, focused dataflow as the visual programming paradigm and was gently loaned by a group of two students from Departamento de Informática at Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa. ArduinoFlow was developed with the same technology as Visualino (Eugenia and Eclipse GMF), so usability aspects can be compared and identified in detail. This new programming paradigm can evaluate the sequential logical thinking against several flows of code (or blocks).

Those three languages that participate in the experiments have its own questionnaire. Thus, it is possible to determine misunderstood concepts, figures or icons for each one of them. The questionnaires are composed by a series of questions and open commentaries to acquire informal content like suggestions.

Identifying the user's characteristics, is another important task in the evaluation process. Questionnaires, interviews and real environment observation are techniques that help collecting user's information before and during the exam. This information gives deeper knowledge on the language's target users. An exhaustive evaluation can be very expensive, so this experiment should focus in the most critical concepts and activities. Those activities represent real programs and behaviors that robots like this should have. Children's classes take most of their time, and it is difficult to appoint test case sessions. Luckily, two schools accepted the invitation for the study case.

7.2 Identify Visualino's goals

As previously defined, Visualino will be evaluated through an experiment where it is tried to answer the following:

- Is Visualino more effective in creating new behaviors than Lego MindStorms?
- Is Visualino more efficient in programming new behaviors than Lego MindStorms?

- Are participants using Visualino more confident by their performance than when using Lego MindStorms language?

The goal is to analyze the performance of its users, comparing it with an existing alternative (Lego MindStorms), taking into consideration factors like efficiency, effectiveness, apprenticeship and confidence. The following hypotheses were formulated:

- H1null There is no significant difference in the effectiveness of the participants' program when using Visualino vs. when using Lego MindStorms.
- H1alt Using Visualino increases the effectiveness of the user's program over the use of Lego MindStorms.
- H2null There is no significant difference in the efficiency of the participants' program when using Visualino vs. when using Lego MindStorms.
- H2alt Using Visualino increases the efficiency of the user's program over the use of Lego MindStorms.
- H3null There is no significant impact in the user's satisfaction to program when using Visualino vs. when using Lego MindStorms.
- H3alt Using Visualino increases the user's satisfaction to program over the use of Lego MindStorms.

7.3 Experiments general procedure

There were two experiments to evaluate Visualino language. Both held similar processes, in order to achieve results that could answer the formulated hypothesis and goals. A consent form was sent to the parents or responsible adults of children who participated in the experiments so we could formally study Visualino usability with its target users. As provided in the consent form, the information collected within these experiments is presented anonymously.

The evaluator introduced himself as a teacher for robot languages, and hid the experiment's goals from children, treating the languages in a similar way – Visualino was not presented as the main language to be tested, and the evaluator did not pass the notion to children that Visualino was developed by him; this could possible damage the results obtained from the experiments.

The first study was conducted in the context of a primary school, named Colégio Campo de Flores[1]. The study's goal was to identify key factors that would determine Visualino's interactivity and usability against Lego MindStorms with eight-year-old children. The experiment was composed of:

[1] http://campodeflores.com/blog/2013/07/29/alunos-do-3o-ano-do-colegio-campo-de-flores-participaram-num-estudo-de-adequacao-e-qualidade-de-uma-linguagem-de-programacao/

- Twelve students that were randomly separated in two groups;
- Each group had six students that were organized in three teams;
- The teams were arranged with two students that would be tested at the same time;
- Children were eight years old and about to finish the third grade.

This information is shown in table **??**.

The participants recruited to test both languages, were composed in two groups. Since all of them belong to the same school and environment, they were treated equally despite specific individual characteristics.

Table 7.1: First experiment evaluation

	First Group	Second Group
Teams	Three teams	Three teams
Each Team	2 students	2 students
Age	8 years old	8 years old
Tested Languages	Visualino and Lego MindStorms	Lego MindStorms and Visualino

The second experiment was performed in a day-school named Natel which also gathers students from high schools. This experiment was conducted to evaluate Visualino usability with children at different ages. It had Lego MindStorms language process evaluation as well, plus the ArduinoFlow language. An experiment was made to test this new language in order to minimize bias from the results relative to the workbench usability (Visualino and ArduinoFlow were implemented and designed in the same technology, Eugenia/Eclipse). The study contained the following subjects and groups:

- Ten students were separated into five teams, according to their age;
- The first team had two twelve-year-old students;
- Two ten-year-old children composed the second team;
- Third, fourth and fifth teams contained two eight-year-old child each;
- The first three teams participated in Lego and Visualino languages;
- Children from third, fourth and fifth teams tested ArduinoFlow language;
- Once again, each team contained two students that were tested at the same time.

This information is shown in table 7.2.

Table 7.2: Second experiment evaluation

	First Team	Second Team	Third Team	Fourth Team	Fifth Team
Age	12 years old	10 years old	8 years old	8 years old	8 years old
Tested Languages	Visualino and Lego MindStorms	Visualino and Lego MindStorms	ArduinoFlow, Visualino and Lego MindStorms	ArduinoFlow and Visualino	ArduinoFlow and Visualino

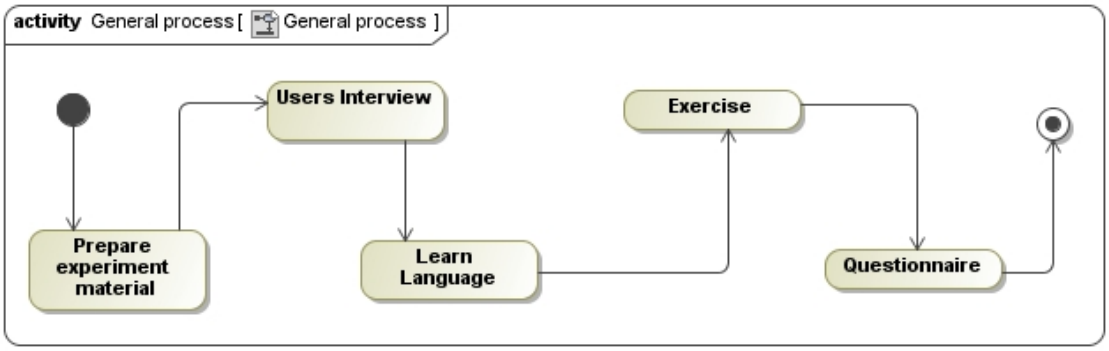


Figure 7.1: General process for the DSL evaluation

Both experiments took place in the corresponding school's classrooms and had a similar process, depicted in figure 7.1. The experiments also had differences, such as subjects characteristics (e.g., age and academic year) and the languages tested. After preparing the classroom with the necessary equipment, the recruited subjects showed up at the class and were interviewed so we could gather relevant information about them. After collecting this specific data, the robot (Lego or Farrusco) and its language platform, were presented, starting with its physical components together with the specific language concepts. At this phase, children practiced some examples using the introduced blocks, and actually observing the developed behavior being executed by the robot.

An exercise split in four sections was presented to children after they experienced the language concepts and its workbench. A software tool recorded the desktop environment in order to analyze children's solutions and errors. The evaluation was closed with a questionnaire that would evaluate user satisfaction towards the specific language. The following section shows the prepared material and documentation used for each experiment.

7.4 Prepared Material for the experiments

Before the live experiments, it was necessary to prepare materials that would be used with recruited subjects. Slides containing information about each language, the exercises that composed the evaluation project for children and the instruments used in the experiments are presented in this section.

7.4.1 Slides

Since children do not feel thrilled towards a large extension of documentation, a couple of slides containing languages concepts were prepared to solve this problem. Thus, if they have a problem with any block or concept, they could rapidly consult the respective slide. Each language had its own informative slides, containing the following structure:

- The target robot (Lego or Farrusco) components and their functionality;
- An introduction containing the different blocks and examples that compose a program in that specific Language;
- Real examples that children should understand and practice;
- A final exercise that children should think and develop at a specific time;
- One last slide containing acknowledgments and presenting children the opportunity to develop their own program.

Lego and Farrusco robots present similar components and concepts, so the developed slides for Visualino and Lego MindStorms languages are equivalent to each other. These slides, as the experiments and exercises, are separated in three major programming domain concepts.

- Children need to be familiarized with actuate blocks, given that these are the simpler ones, and convey the base concepts of the language. This is done by setting a goal of having them developing an example that involves the robot doing a sequence of actions, e.g. activate the motors, wait a couple of seconds, turn on the LED (for Farrusco) or play a sound (for Lego robot).
- This follows with another concept, to have actions done at the same time, e.g. the robot moves forward while blinking the LED (for Farrusco) or playing two different sounds (for Lego). This concept was not tested in ArduinoFlow language, because it was not implemented.
- Finally, the major and hardest challenge is to implement a condition that would make the robot smarter (e.g., if the robot collides with its collision sensors, it should have a different trajectory). This last concept involves the robot's sensors, which represent conditions as an if-then-else chain. The projected exercise consists in these three concepts, which will be used in the last part of the evaluation sessions.

7.4.2 Exercises

These exercises will evaluate the language concepts and usability towards the recruited subjects. Each language has its own exercise, and this section encompasses each one of those.

The exercise purpose is to evaluate children's abilities to create one complex robot behavior through the concepts and examples that they have experienced before. They start with simple tasks very similar with the previous examples, attempting to combine and associate the different behaviors into a complex one. This exercise will evaluate if the tested users were able to accomplish each step of the exercise and how good they were while performing such tasks.

7.4.2.1 Visualino

The designed project is composed of four parts:

- The first exercise consisted of creating a simple sequence of actions. This exercise can evaluate the sequence concept and other two actuate blocks. Those are the wait-time and LED blocks. Children should tell the robot to blink its LED each two seconds.

- The following exercise consisted in creating a complex program, making use of the solution previously found. Now, they should tell the robot to move forward and at the same time, blinking its LED as before. The power of parallel execution is performed by the Parallel block. The motors block can activate the motors from the robot. Those are the two new concepts, which are evaluated in this exercise.

- The third exercise's objective measured the children accuracy in adding a condition to the robot actions. Therefore, the robot should continue blinking its LED and moving forward, only if it could not detect any obstacle. If it detected an obstacle, it should move backwards for 2 seconds, and then continue to move forward. Here a new control node is expected to be on the program, the Decider, such as the Infrared block.

- Adding a different condition to the robot is the last task of the exercise. The program should remain the same as before, but now if the robot could not detect an obstacle with its Infrared block, it should be aware of its crash sensors. If the robot collides with an obstacle, it should move backwards, once again for 2 seconds.

The solution for this exercise is depicted in figure 7.2.

7.4.2.2 Lego MindStorms

The next exercise took care of the Lego MindStorms' robot and its software tool. This new project was obligated to bring forward similar exercises to Visualino. In this way, we

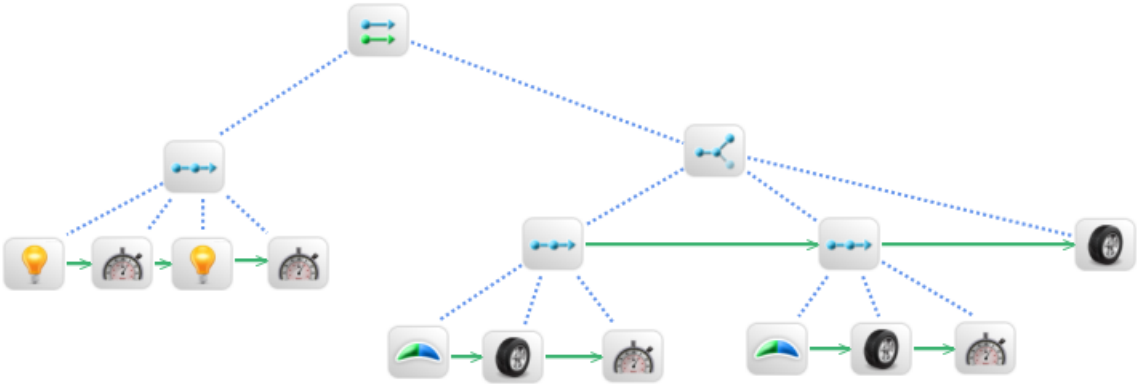


Figure 7.2: Solution for the Visualino exercise

could determine if the possible problems found on Visualino were difficulties regarding the language, or the exercises itself. The Lego project is described in the following lines:

- The first task will test if children can elaborate a simple sequence of actions, as Visualino. So, they are asked to make the robot play two different phrases, and repeat them every two seconds. Once again, a sequence of actions containing wait-time and play sound blocks is evaluated in this task;
- The second task, as Visualino project, children were asked if they can order the robot to move forward and play a specific sound at the same time. The robot should execute the previous example as well;
- The third exercise consists in applying a condition to the Lego robot to the previous exercise. If the robot could detect an obstacle, by its distance sensors, it should move backwards for 2 seconds. If this does not happen, the Lego should continue moving forward;
- For the fourth exercise, a new condition should be added to the robot behavior. Now the robot should be more alert to its moving area. If Lego could not detect any obstacle by the distance sensors, it should be alert of its crashing sensor. If this sensor were bumped with some object, Lego should move backwards once again for two seconds.

The solution for this exercise is depicted in figure 7.3.

7.4.2.3 ArduinoFlow

The final exercise belonged to ArduinoFlow language. This one was designed for only one experiment but it requires the similar behaviors as the other languages' exercises.

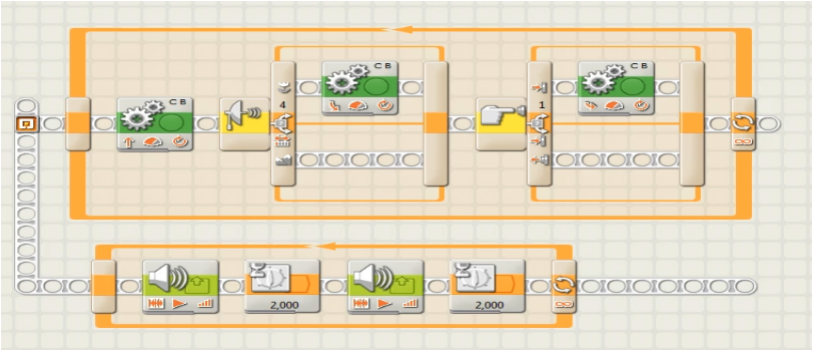


Figure 7.3: Solution for the Lego Mindstorms exercise

The exercise is also separated in four parts:

- To begin the exercise, Farrusco was supposed to blink its LED every two seconds. Children had the liberty to choose the LED's color;
- In the second task, Farrusco should blink its LED once and then it should start to move forward;
- For the third task, children were asked to add a condition to Farrusco. If it could detect an obstacle through its distance sensor, it should go backwards, for 1 second.
- The final task, as for the previous languages, Farrusco should be aware of its collision sensors. If it could not see any object, but bumped with an obstacle, Farrusco should go backward for one second.

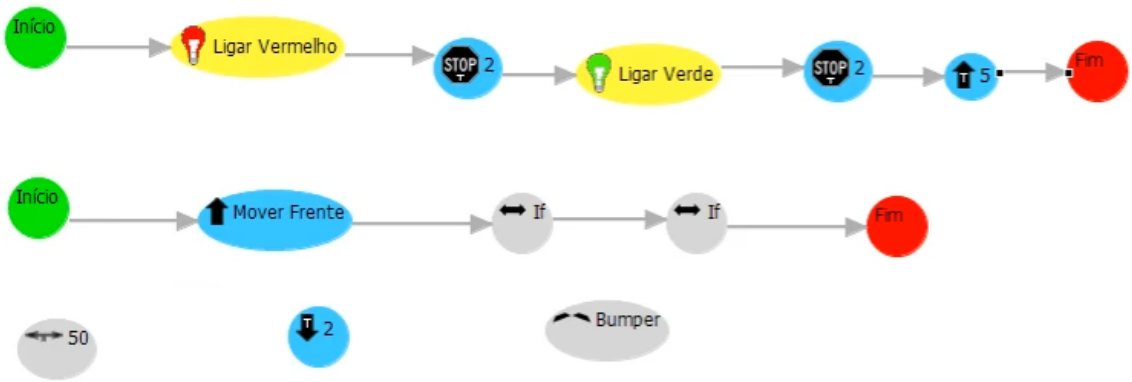


Figure 7.4: Solution for the ArduinoFlow exercise

This exercise has the solution presented in figure 7.4. ArduinoFlow was implemented in Portuguese language, so the solution is also presented in Portuguese.

7.4.3 Questionnaires

A series of questions compose each language's questionnaire. To avoid untrustworthy data, questionnaires should be small and quick to do, so children could answer correctly and it could be possible to dodge some random answers. Multiple-choice questions have their own scale, based on iconic figures. Visualino, Lego MindStorms and ArduinoFlow had similar questionnaires that focused the same kind of questions and answers. The real questionnaires containing the actual questions and figures used are shown in the following pages. The questionnaire regarding each one of the languages consisted in the following queries:

- The first question asks if the user enjoyed the class. In this way it is possible to measure if the user felt uninterested in class.

- The second question lists every block studied at class, and asks to rate their corresponding difficulty. This question is required to study which concepts/blocks are hard to understand and if those correspond to the ones present in the user mental model.

- The third question aims at getting feedback about the icons used for each block. In this way, the user could refer blocks' icons he did not like. This was an open question so the user could mention any figure or simple just an idea to improve the icon's readability.

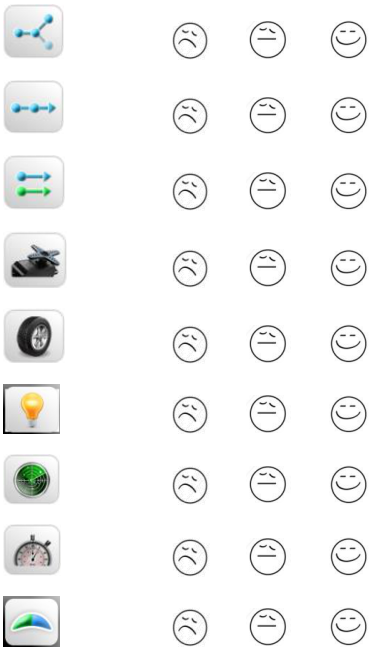
- The last question had once again the presented iconic scale, and it enquires about the confidence level from the user to program a behavior of his own.

Figure 7.5 contains the Visualino individual questionnaire. Figure 7.6 shows the Lego MindStorms questionnaire. Figure 7.7 contains the ArduinoFlow individual questionnaire.

Questionnaire

Did you like the robotic classes?

Which of the blocks did you find most difficult to use?



Would you change any of these figures? Which of them?

_____

Why?

_____

_____

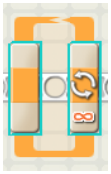
Do you think you can program your own behaviors on the Visualino language?

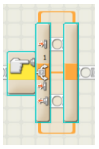
Figure 7.5: Visualino Questionnaire

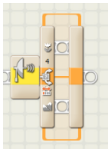
Questionnaire

Did you like the robotic classes?

Which of the blocks did you find most difficult to use?







Would you change any of these figures? Which of them?

_____

Why?

_____

_____

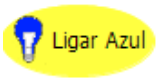
Do you think you can program your own behaviors on the Lego language?

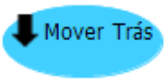
Figure 7.6: Lego Questionnaire

Questionnaire

Did you enjoy the robotic classes?

What did you think it was the hardest thing to do in these blocks that you learned:

- Change Farrusco's lights? 

- Put Farrusco on motion? 

- Tilt Farrusco's head? 

- What about conditions?
  - "If" block? 

  - Bumpers? 

  - Distance sensor? 

For all those blocks that you think are the hardest, explain me why you think that?

_____

_____

_____

Would you change anything in these figures? Which ones?

_____

Do you think you can program your own behaviors in this tool?

Figure 7.7: ArduinoFlow Questionnaire

A new questionnaire was made to compare Lego MindStorms and Visualino languages. With this questionnaire, it is possible to get some comparative data about each language's friendliness towards the users.

- The intention behind the first question was to acquire which of the languages the user enjoyed the most. In his answer, the user selects one of two possible options: Lego MindStorms or Visualino. The results from this question do not imply that the language with more votes is easier than the other; it merely means that users felt that it was the friendliest.

- In order to attain which of the languages was easier to use a second question is formulated: which of the languages was the more difficult to interact with. The answers to this question allow the identification of the easier language to use.

- The third and fourth questions aim to understand why the users gave the previous answer. This kind of feedback is important, as it allows a deeper understanding behind the choices made in the previous question. The former's goal was to identify the reasons that were behind the language being difficult to use, and the goal for the latter is to receive suggestions on how the language could be improved, making it easier to use. The user's mental model can be traced with these open commentaries, leading to possible future improvements of the language.

- The questionnaire ends with three multiple-choice questions, targeting concepts present in both languages, where the user selected – for the concepts listed below – which language better conveyed each concept.

  - Program a simple sequence of actions, using any action blocks;

  - Develop a behavior which the robot should activate more than one actuator component at the same-time;

  - Improve the Robot's behavior through its sensors. It should trigger an action when a sensor is activated.

Figure 7.8 contains the comparative questionnaire regarding Visualino and Lego MindStorms.

Lego and Farrusco

In these classes you tried two robots. Each one had its own tool platform. Which one did you enjoy the most?

Farrusco ☐ **Lego** ☐

Which one was the hardest?

Farrusco ☐ **Lego** ☐

Why?

_____

_____

What would you change in the tool that you enjoyed less?

_____

_____

Think that we wanted to put Lego or Farrusco to do a sequence of actions. For example walk straight ahead, turn on the light (or speak) and then stop. In which of these two could you do it better?

Farrusco ☐ **Lego** ☐

And if we wanted Lego or Farrusco to do different things at the same time. For example, turn on the lights (on Farrusco) / speak (on Lego) while walking forward. To program this behavior, are you more at ease on which one?

Farrusco ☐ **Lego** ☐

IF Farrusco or Lego hits an obstacle or sees anything on the way, it should move backwards. Which of these two would be easier to program this behavior?

Farrusco ☐ **Lego** ☐

Figure 7.8: Comparative Questionnaire

Table 7.3: Hardware characteristics used in the experiments

Device Type	Component	Detailed Information
Desktop Computer	Clock Speed	2.2 to 3.0 GHz
	Cores	2to 8 cores
	Ram	2 to 16 GB
	Hard Disk Space	320GB to 1 TB
	Network	Wired
	Monitor	17" to 24"
	Power	200 - 500 W
	USB Ports	4 to 8 ports
	Mouse	wired
	Keyboard	wired
Laptop Computer	Clock Speed	1.6 to 2.2 GHz
	Cores	2 to 8 cores
	Ram	2 to 8 GB
	Hard Disk Space	320GB to 1 TB
	Network	Wired
	Monitor	13.3 to 15.6 inches
	Power	40 - 120 W
	USB Ports	2 to 4 ports
	Touchpad	Built-in
	Mouse	wired
	Keyboard	Built-in

7.4.4 Working equipment and environment

The experiments were made at different schools; however, they required the same equipment and a stable work environment. This section has the objective to describe equipment and work area characteristics.

It is important to document characteristics and specific details that were present in the experiments, so it could be possible to identify inappropriate equipment or working environment that could damage the results from the experience. Since the experience involve software installed on computers, it is essential to describe computers hardware and software details [40].

Therefore, table 7.3 describes the computers used in the experiments. Both laptop and desktop computers are analyzed.

Table 7.4 describes the software installed, used to develop behaviors for each robot. Lego MindStorms and Eclipse/Java environment from Visualino and ArduinoFlow are mentioned.

The working environment prepared for the user, can also influence usability results. The main environment equipment was composed by:

- One Working desk per group;
- One Chair per child;

Table 7.4: Software programs used in the experiments

Operation System Environment	Working Platforms
Operating System	Microsoft Windows 7
Working Platforms	Java version 1.7.0
	Eclipse Juno version 4
Languages Workbenches	Visualino Workbench
	ArduinoFlow Workbench
	Lego MindStorms NXT 2.0 Programming software
	Arduino IDE for compiling code

- Windows which provided light to the classroom;
- Office lights were present as well;
- The rooms were prepared with Aircondition system.

The robots used to execute the programs developed by the users have its own details. Table 7.5 describes the details that compose the components of each robot.

Table 7.5: Robots Characteristics

Robot	Hardware	Details
Farrusco with Arduino Board	Duemilanove Board	ATMega328
	Operating Voltage	5 V
	Clock Speed	16 MHz
	Digital I/O Pins	14 pins
	Analog Input Pins	6 pins
	Infrared Sensor	1 unit
	Collision Sensor	2 units
	LED	1 unit
	Servo motor	1 unit
	DC motors	2 units
Lego MindStorms	Version	NXT 2.0
	Input ports	4
	Output ports	3
	Batteries	6 AA bateries
	Distance sensor	1 unit
	Infrared Sensor	1 unit
	Collision Sensor	1 unit
	Claw	2 units
	Motor	1 unit
	Loudspeaker	1 unit
	Screen	100x64 pixel LCD

7.5 Experiments Process

This section shows the specific procedures taken for each experiment involving the technologies and the prepared material previously mentioned.

7.5.1 Pilot Session

To avoid possible problems that could happen at a real evaluation session, it is important to practice those lessons with the respective prepared material and subjects. In this case, it was possible to recruit a child that fulfilled the requirements of the Visualino's target users. With his help, it was possible to find some errors present in the documentation and even language's slips that would be hard to find without this pilot session. Pilot testing allowed measuring the time spent for a session, which contained a similar process for the real test case. Through this session it was possible to confirm if the created exercises were feasible and clear to the child.

This informal session counted only with Visualino, considering that the other languages counted with similar documentation and software requirements.

7.5.2 The First experiment

As previously said, the first experiment encompassed twelve students at the third grade from the same school. The experiment took place in a primary and secondary school named Colégio Campo de Flores and was conducted in Portuguese language. Since this experiment's goal was to compare Visualino usability with Lego MindStorms, the recruited subjects were randomly separated in two groups as presented in figure 7.9. Each group had six students, which were distributed in teams of two subjects, working together in the same computer and desk.

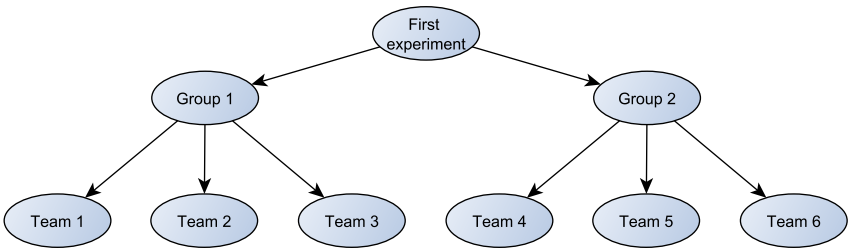


Figure 7.9: Groups that composed the first experiment

The process taken for each group is presented in figure 7.10. It started with an interview, so we could identify personal characteristics (specifically, those mentioned in the user profile section) from the recruited subjects.

The first group started with Visualino Language and the other with Lego MindStorms language. If both groups started with Lego MindStorms language, probably they will be better on the second language and its corresponding test. The same idea is applied if

they start with Visualino and then Lego MindStorms. Children would have more experience on the second language, because they learned some new concepts and a new logical thinking to program robots, which they did not have before. Conversely, it could also have a negative outcome, considering that children could be confused with concepts learned from the previous language. With this technique, it is easier to analyze which concepts and characteristics were hard to understand by children, and if those problems came from the language learned on the first class or if it was certainly usability issues. Thus, some possible contamination on the final results is avoided with this method.

The learning phase encompassed an introduction to the target robot components and behaviors. Children could practice some examples in the language platform, upload and observe their program being executed by the robot. The exercise phase included a project that should be developed by each team without any help from the other teams or the evaluator. One questionnaire regarding the first language was presented after children experienced the first exercise. The same is applied for the second language, plus an additional questionnaire, which children were asked to compare both languages.

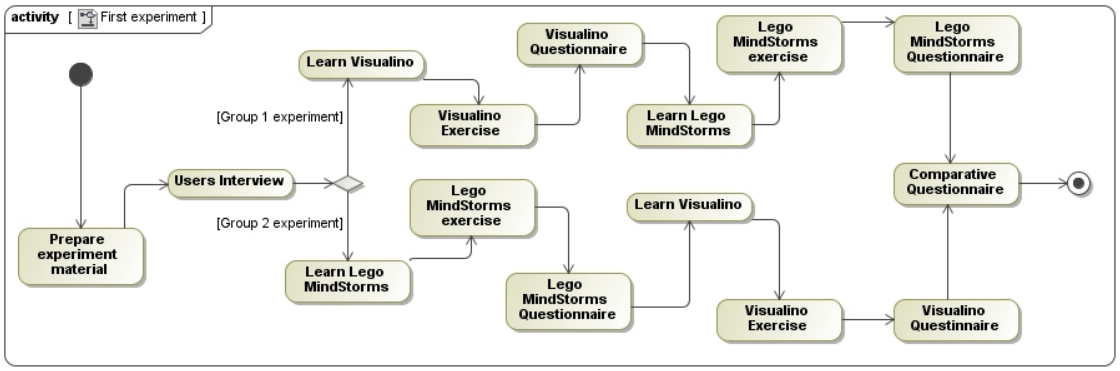


Figure 7.10: Process taken for each group

The equipment presented in section 7.4.4 was prepared before the recruited subjects entered in the working classroom. After these preparations, children were interviewed in order to know more about them so it could be possible to identify their user profile and relevant characteristics. Table 7.6 presents pertinent aspects from the recruited subjects:

Both groups presented frequent contact with technologies such as computers, tablets and video game consoles. However, none of them experienced any computer programming language before neither robot programming tools. Group 1 encompassed two female and four male students. Group 2 was composed by other six students – three girls and three boys. The average grades for Portuguese, Mathematic and Science classes are described in table 7.7. They are generally good students.

Table 7.6: User Profile

Personal characteristics	Age	8 years old
	Academic year	Third grade
	Native Language	Portuguese
Technological Background	Video game consoles	Frequent Use
	Tablets use	Frequent
	Computer Knowledge	Beginners
	Previous Robotic experiences	None
	Programming skills	None

Table 7.7: Characteristics from each group

	Gender (Male/Female)	Portuguese Grade (1 to 5)	Mathematic Grade (1 to 5)	Science Grade (1 to 5)
Group 1	2 F, 4 M	4	4	5
Group 2	3 F, 3 M	4	4	5

7.5.3 The Second experiment

The second experiment was composed by ten children from one day-school named Natel. Once again, this experiment was conducted in Portuguese, at one school's classroom. This experiment's objective was to test Visualino usability with children from eight to twelve years old. Children also tested Lego MindStorms, and ArduinoFlow language. Concerning Group 1, students were separated by age for each team. Group 2 counted only with eight years old children. Figure 7.11 shows the groups that participated in this experiment. Team 3 tested Visualino, Lego MindStorms and ArduinoFlow, and that is why they belong to group 2 as well.

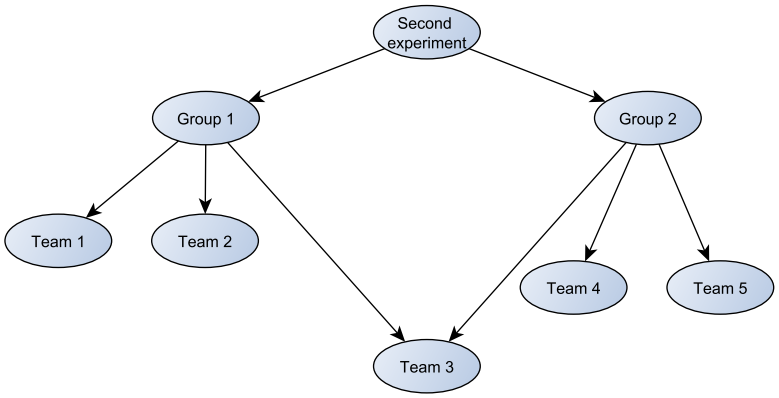


Figure 7.11: Groups that composed the second experiment

Group 1 had a similar evaluation process regarding the first experiment. After setting up the environment, with the prepared material and instruments, an interview took place in order to understand and identify their characteristics. Those characteristics are

described in Tables 7.8 and 7.9. Team 1 were composed by two male twelve years-old subjects and currently in the seventh grade. Those were generally good students both in mathematics, science and in their native language, Portuguese. Team 2 were composed by two female students, in the fifth grade. Unlike the other teams, these two students had lower grades at mathematics, but were generally good students on the other disciplines. One boy and a girl composed Team 3 – they were eight years-old students in the third grade and presented generally good grades. Older students (12 year-old) exhibited more practice and a frequent use with technologies, such as computer games and web browsing. However, video games is still the most preferred hobby for these older students. Younger students from Team 2 and 3, had contact with technology as well, but not so frequent as the older students.

Table 7.8: Group 1 User Profile

	Age	School grade	Gender (Male/Female)	Portuguese Grade (1 to 5)	Mathematic Grade (1 to 5)	Science Grade (1 to 5)
Team 1	12 years old	Seventh school year	2 M	4	4	4
Team 2	10 years old	Fifth school year	2 F	4	3	4
Team 3	8 years old	Third grade	1 F, 1 M	4	4	5

Table 7.9: Technological background from group 1

	Team 1	Team 2	Team 3
Video game consoles	Very Frequent	Frequent	Frequent
Tablets use	Frequent	Frequent	Frequent
Computer Knowledge	Intermediate	Beginners	Beginners
Previous robotic experiences	None	None	None
Programming skills	None	None	None

Group 1 started to learn Visualino concepts along with Farrusco components. As the previous experiment, children from this group were asked to practice some examples in order to understand the language concepts and semantics; they also observed Farrusco executing the developed behaviors during this phase. The second phase consisted in testing the learned concepts, by having the children do the specific designed exercises presented in section 7.4.2.1. Once they finished the exercise, they were asked to fill a questionnaire regarding Visualino language.

The second session contained Lego MindStorms being tested with the same children from Group 1. It has a similar process as Visualino, containing the learning and the

exercise phase. After the exercise phase was completed, they were asked to fill the questionnaire for Lego, along with the comparative questionnaire 7.4.3.

Group 2 was composed by eight years old children in the third grade. They tested Visualino and ArduinoFlow language. The test was important because it could minimize the bias from the results relative to workbenches' usability. This particular phase of the experiment, held similar aspects from the other phases. It started with an interview with the children elected for this experiment. Pertinent characteristics are presented in tables 7.10 and 7.11. Team 3 characteristics had changed, considering they already tested Visualino and Lego MindStorms languages. Team 4 was composed by two female students such as Team 5 – both teams had students with relative weak grades at mathematics but good in the other courses; Video games were an occasional hobby of Team 3 and 4. Yet, their activities were occupied by tablets and computer games.

Table 7.10: Group 2 User Profile

	Age	Grade	Gender (Male/Female)	Portuguese Grade (1 to 5)	Mathemati Grade (1 to 5)	Science Grade (1 to 5)
Team 3	8 years old	Third grade	1 F, 1 M	4	4	5
Team 4	8 years old	Third grade	2 F	4	3	4
Team 5	8 years old	Third grade	2 F	4	3	4

Table 7.11: Technological background from group 2

	Team 3	Team 4	Team 5
Video game consoles	Frequent	Occasional	Occasional
Tablets use	Frequent	Frequent	Frequent
Computer Knowledge	Beginners	Beginners	Beginners
Previous robotic experiences	Farrusco (Visualino) and Lego MindStorms	None	None
Programming skills	Lego MindStorms and Visualino	None	None

The process maintained its structure, followed by the learning and exercise phase. Children from group 2 had to complete the questionnaire from ArduinoFlow language. The process taken for group 2 experiment is presented in figure 7.12.

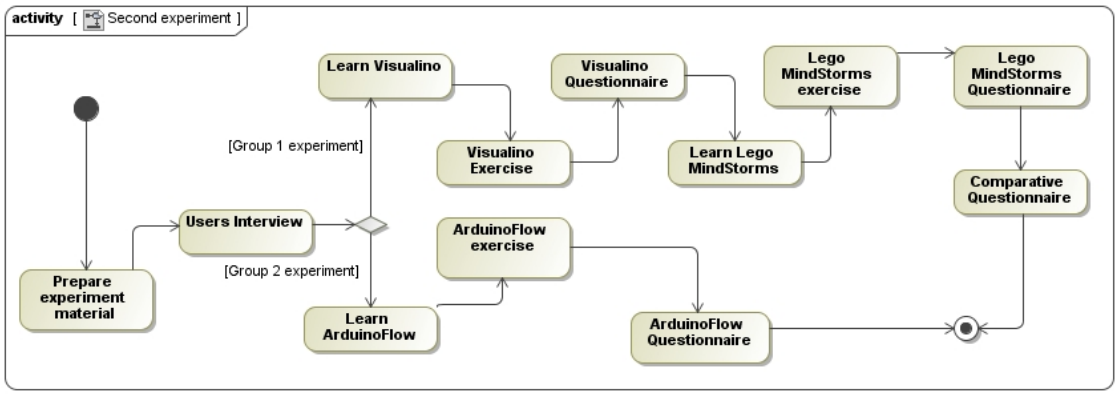


Figure 7.12: Process taken for each group

8

Results Analysis

The achieved results from each experiment are discussed in this section. Related questionnaires' information and observations taken during the experiments are shown as well. This section is separated in two subsections. The first experiment presents the results for the comparative study between Lego MindStorms and Visualino with eight-year old children. The second experiment shows the results for Lego MindStorms and Visualino regarding subjects of different ages. Usability observations from ArduinoFlow are shown as well.

8.1 Results from the first experiment

Section 7.2 referred the main goals for Visualino. Through this experiment it is possible to answer the formulated questions and hypotheses made in the aforementioned section.

Learning times

The steep learning curve can be measured through the time spent in the learning phases for each language. Learning phase encompassed the study of the specific robot components, the language concepts and experiencing the provided examples. Tables 8.1 and 8.2 encompass the time spent in the learning phase for Group 1 – the first table describes the first language learned in Group 1, being Visualino, and the second table represents the Lego MindStorms case, once again related to Group 1. The time children spent to observe the robot performing the developed behavior are not considered in this case, as the time for compiling and uploading the program. Instead, these tasks are related to the time children needed to understand and practice the examples displayed in the prepared

slides.

Table 8.1: Group 1 time spent for each task example in Visualino

	Sequential Actions (LED + Motors)	Parallel Actions (LED + Motors)	Collision Sensor	Distance Sensor
Time	17 minutes	14 minutes	14 minutes	8 minutes

Table 8.2: Group 1 time spent for each task example in Lego MindStorms

	Sequential Actions (Loudspeaker + Motors)	Parallel Actions with Loop concept	Collision Sensor	Distance Sensor
Time	10 minutes	8 minutes	6 minutes	4 minutes

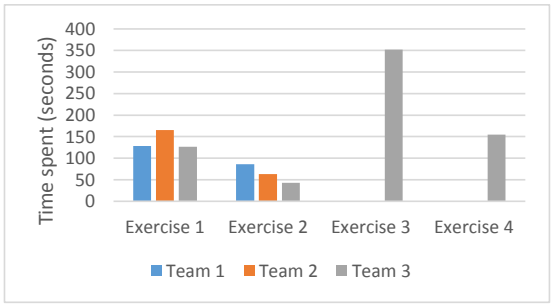
Table 8.3: Group 2 time spent for each task example in Lego MindStorms

	Sequential Actions (Loudspeaker + Motors)	Parallel Actions with Loop concept	Collision Sensor	Distance Sensor
Time	14 minutes	12 minutes	9 minutes	7 minutes

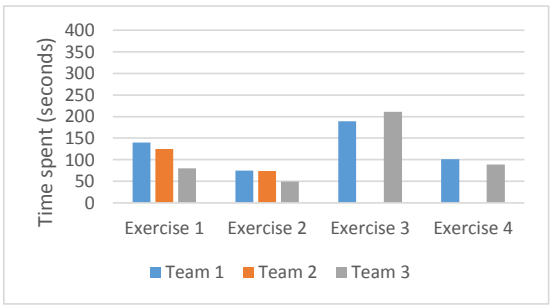
Table 8.4: Group 2 time spent for each task example in Visualino

	Sequential Actions (LED + Motors)	Parallel Actions (LED + Motors)	Collision Sensor	Distance Sensor
Time	13 minutes	9 minutes	8 minutes	4 minutes

Visualino has a higher learning curve, because children took longer to complete the examples and understand the behavior behind them. This is perceptible in the first task, where children need to develop a sequence of actions involving the Farrusco's LED and its motors. They should be aware of the two links available (brother and son) that are needed to produce a well-formed tree. Understanding the control nodes (Parallel, Sequential and Decider) is an additional task in Visualino that increases the time spent in this learning process, although sequential and parallel nodes are easy to learn by children. Parallel actions are perceptible as equally fast to learn in Visualino and Lego MindStorms, although there is irrelevant difference of two minutes between both languages. Both condition tasks (Collision and Distance sensor scenarios) took less time to learn in Lego MindStorms, essentially because this behavior is made through only three steps/blocks, while in Visualino is necessary to build the tree with the decider node and sequential nodes for each conditional case.



(a) Group 1 time spent for Visualino



(b) Group 1 time spent for Lego

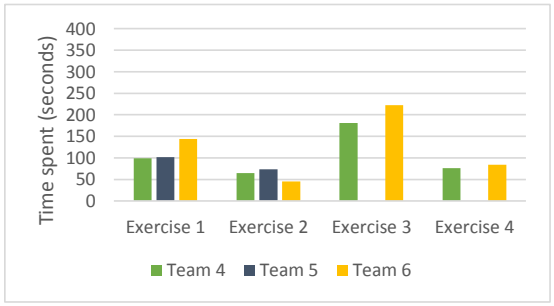
Figure 8.1: Group 1 time spent in the exercises for each language – started with Visualino

Visualino and Lego MindStorms exercises results

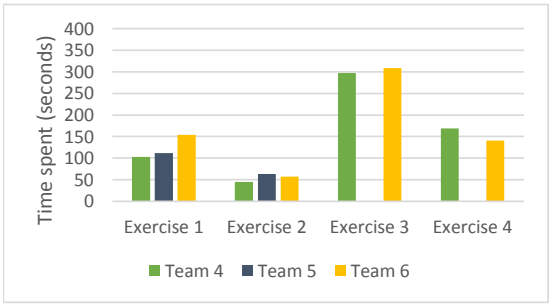
Charts from figures 8.1 evaluate the Visualino's efficiency and effectiveness against Lego MindStorms language, for Group 1. The first two exercises – simple tasks – which encompass the creation of a sequence of actions and a parallel behavior, were easily completed by each team for both languages. Children took similar time periods to complete those tasks for both languages.

Visualino's third and fourth exercises – complex tasks – were only completed by Team 3, and it took longer than the Lego MindStorms' exercises.

Although Team 1 could not complete the complex tasks from Visualino, they were able to finish the Lego MindStorms exercises. Those tasks which were not completed, contained wrong solutions that would result in an erroneous behavior executed by the robot. Team 2 could not complete the complex tasks neither for Visualino nor Lego MindStorms.



(a) Group 2 time spent for Lego



(b) Group 2 time spent for Visualino

Figure 8.2: Group 2 time spent in the exercises for each language – started with Lego MindStorms

Figure 8.1 depicts two graphs corresponding to Group 2 time spent in the exercises for Visualino and Lego MindStorms languages. Once again, children were able to solve the first two exercises with similar times for Visualino and Lego MindStorms. Team 5

children were not able to complete the exercise, since they developed an incorrect solution for the third and fourth exercises – this situation occurred both for Visualino and Lego MindStorms. Although children had started with Lego MindStorms language, they took longer to complete Visualino complex exercises.

Comparing the results from both groups

Group 1 took less time to solve Lego MindStorms' exercise, since they started with Visualino language; Likewise, Group 2 achieved better results in Visualino than Group 1. We can infer that the first-learnt language helps to develop the robot model in the children's mind as the concepts involved in this domain. Although Visualino had poor results when compared to Lego MindStorms, it also helped children from Group 1 to achieve better results in Lego when comparing with children from Group 2.

Effectiveness can be measured through the number of well-completed exercises in both languages. All teams were able to finish the simple exercises for both languages. Lego MindStorms complex tasks were well-completed by four groups while in Visualino only three teams were able to finish the corresponding complex exercises. The common errors section presents the flaws that children made while searching for the solution in these complex tasks, suggesting possible language's misconceived notions on children.

Common errors

Both experiments counted with model errors committed by the recruited subjects. This section presents some of those errors and possibles concepts of the languages that children did not understand.

The third and fourth exercise, which contained the implementation of conditions, were the hardest problems to be developed by children. Figure 8.3 shows a wrong condition implementation.

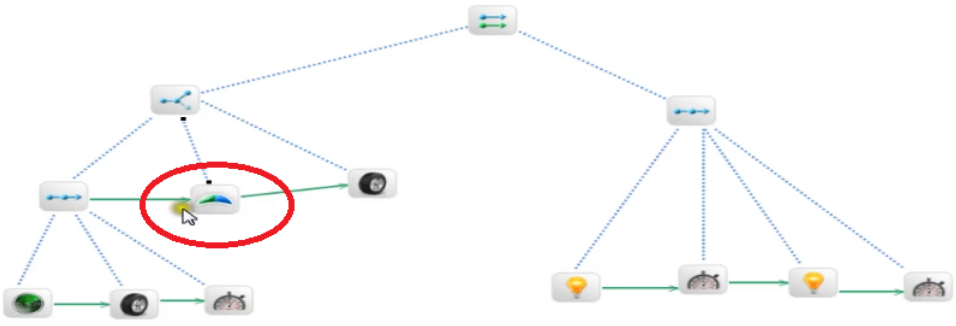


Figure 8.3: User program snapshot containing a condition implementation error

Children often treat the decider node as the node that decides which action to execute. This notion is not wrong; however, they forgot to develop the actions that follow the condition. In Visualino language, an action is frequently implemented with a sequence block

and its corresponding children nodes. The users commonly used the condition blocks (collision and distance sensor), thinking that automatically creates the corresponding reaction in the robot. The children behind the exercise from the previous figure, justified their implementation saying that the bumper node should made Farrusco move backwards once it collides with an obstacle.

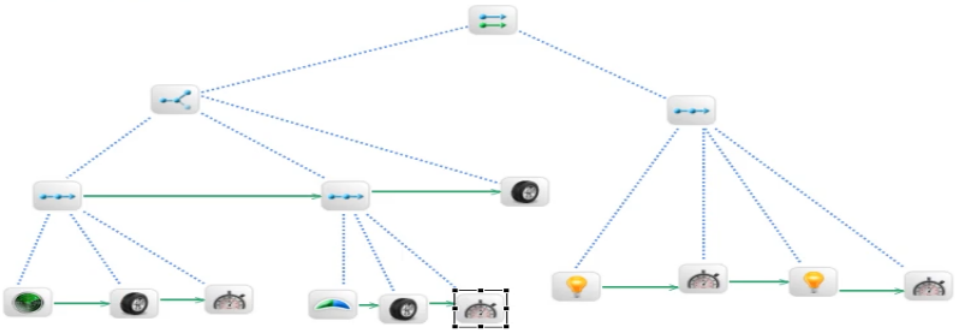


Figure 8.4: User program snapshot containing a correct implemented condition

Children from Team 1 designed the same behavior correctly, as shown in figure 8.4.

Children also developed wrong implementations in Lego MindStorms language. Figure 8.5 shows an implemented condition with the distance sensor. There are three execution flows, where the third one orders the engines to move always forward. This completely cancels the second execution flow, where the condition behavior is implemented – if the robot detects an obstacle it should go backwards for a specific time. This results in an erroneous behavior executed by the Lego robot.

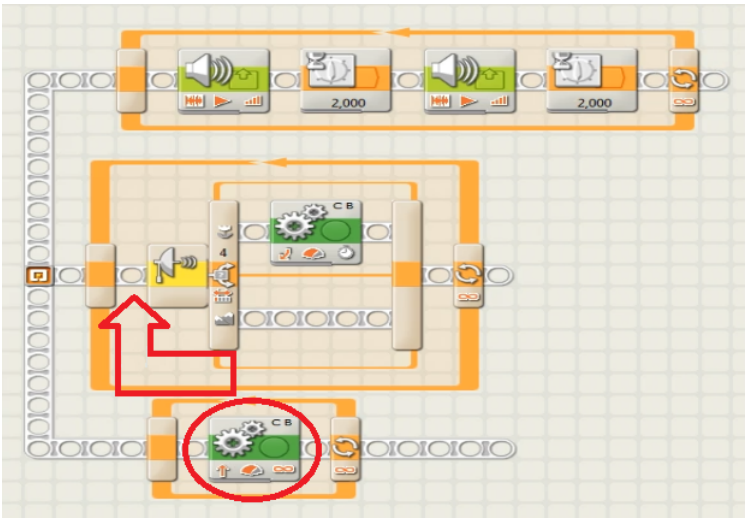


Figure 8.5: User program snapshot containing a condition implementation error in Lego

Children from Team 4 placed the engine block where it belongs – inside the loop of the second execution flow. This behavior, presented in figure 8.6, was the solution for the distance sensor condition problem (exercise 3).

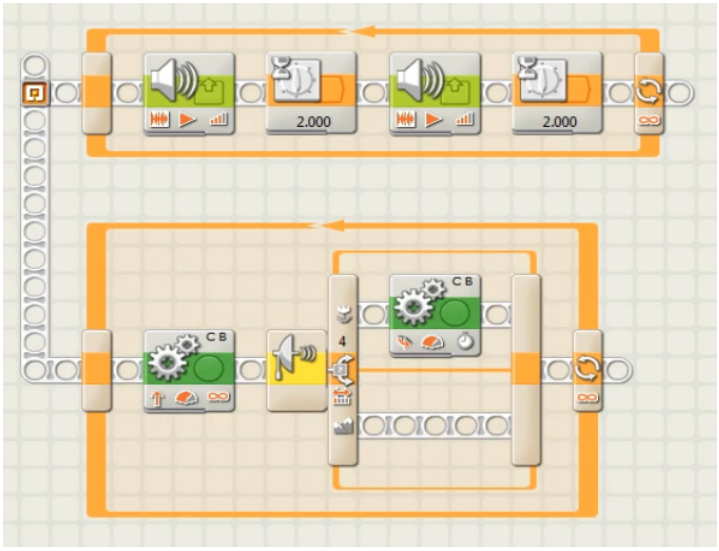
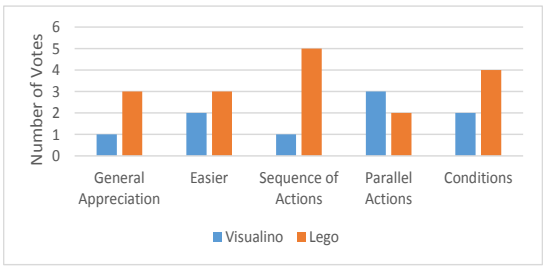


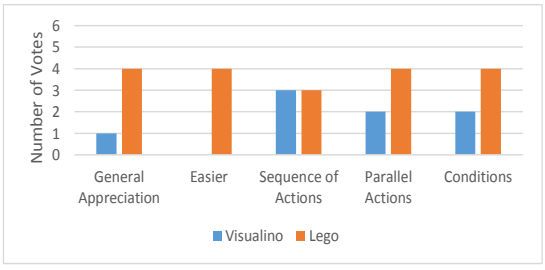
Figure 8.6: User program snapshot containing a well-implemented condition in Lego MindStorms

Satisfaction Level obtained through the Questionnaires

The satisfaction level was measured by the comparative questionnaire presented in section 7.4.3. Charts from the figure 8.7 encompass the chosen answers for the comparative questionnaire made for Lego MindStorms and Visualino. It is important to analyze the chosen answers in each group, considering the order in which children learned both languages.



(a) Group 1 answers – started with Visualino



(b) Group 2 answers – started with Lego

Figure 8.7: Comparative questionnaire answer by each Group

Most subjects from this experiment answered that Lego MindStorms language is more appreciated than Visualino; Some of them did not vote saying that enjoyed working with both languages – two subjects from group 1 and another one from group 2 voted for both languages. Two subjects from group 1 voted that Visualino was easier to learn (even if they took longer to complete the exercises or learn the languages concepts), but Lego had more votes for this topic.

Group 1 subjects were more comfortable to construct sequential programs in Lego MindStorms, whilst Group 2 subjects considered both Lego and Visualino. Time spent

on learning Visualino concepts and practicing the examples took longer than Lego MindStorms – this seems to influence these answers/votes as well, since group 1 took much more time learning Visualino than Lego MindStorms. Results for programming parallel behaviors question seemed to be comparable for both languages. For complex tasks which include conditional behaviors, results from the questionnaires show that Lego MindStorms are better to program this type of behaviors, although some children (two from Group 1 and other two from Group 2) still answered Visualino as the preferable language to program this kind of behaviors.

Graph from figure 8.1 joins the questionnaires answers from both groups. Generally, children reckon that Lego MindStorms was more appreciated and easier to use. When asked which language they prefer to develop parallel behaviors, Visualino and Lego MindStorms share almost the same amount of answers, while in sequence and complex/condition task behaviors, Lego MindStorms is preferred.

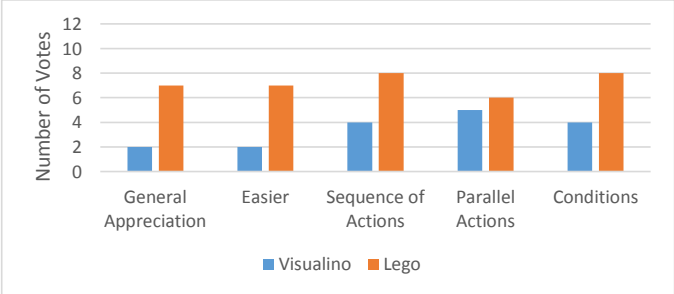


Figure 8.8: Questionnaires answers from both groups

Descriptive statistics

The results obtained from the questionnaires are analyzed in this section. We used the Wilcoxon rank signed test, a non parametric statistical hypothesis test used to compare the two related samples – Visualino against Lego comparative questionnaire answers. It is used considering that the population cannot be assumed to be normally distributed. This test is meant to validate the hypothesis made in section 7.2. For reasons of simplicity, we need to treat all inquiry questions as equally important. The statistic results for the comparative questionnaire regarding the first experiment are shown in figure 8.9.

In this specific case, the Wilcoxon test showed that there is a significant difference between the appreciation scores for the two evaluated languages ($p > .05$, two-tailed test). The ranks table from figure 8.9 clearly shows that Lego MindStorms is the preferred language.

Blocks appreciation

The concrete syntax of Visualino was evaluated through its blocks functionalities. Questionnaires regarding each language provided this information, which is represented in

Descriptive Statistics

	N	Mean	Std. Deviation	Minimum	Maximum
Visualino	5	3,40	1,342	2	5
Lego	5	7,20	,837	6	8

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
Lego - Visualino	Negative Ranks	0[a]	,00	,00
	Positive Ranks	5[b]	3,00	15,00
	Ties	0[c]		
	Total	5		

a. Lego < Visualino

b. Lego > Visualino

c. Lego = Visualino

Test Statistics[a]

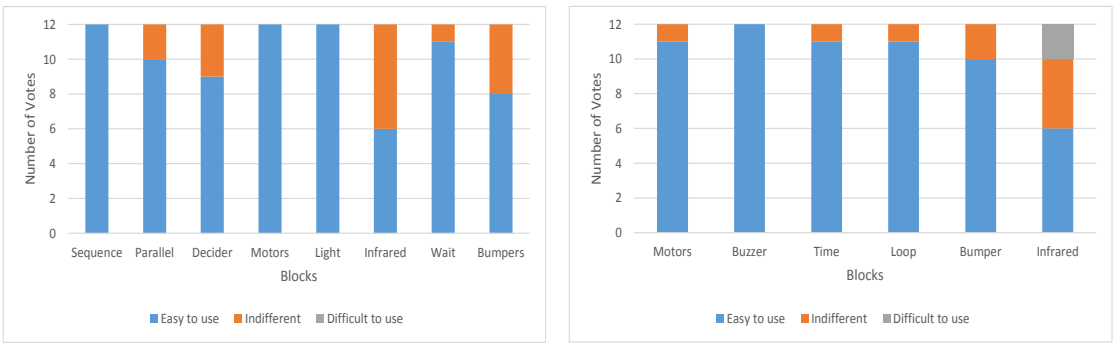
	Lego - Visualino
Z	-2,041[b]
Asymp. Sig. (2-tailed)	,041

a. Wilcoxon Signed Ranks Test

b. Based on negative ranks.

Figure 8.9: Descriptive statistics – Wilcoxson results

figure 8.10. Conditional blocks such as *Collision* and *Infrared* sensors, both for Visualino and Lego MindStorms, appeared to be the most difficult to use, since they were included in the complex tasks from the evaluation exercise for each language. The *Decider* block from Visualino received some negative answers from children, since this one is used in conditional tasks as well.



(a) Visualino's blocks answers (b) Lego MindStorms' blocks answers

Figure 8.10: Answers regarding blocks ease of use for each Language

Experiment's observations

Visualino's blocks were connected to each other through specific links – a low number of users found this feature interesting. Children answered in their questionnaires that the Lego's automatic way of connecting its blocks is a feature that Visualino should have, despite some of them enjoyed the manual connections and the names for it (brother and child).

The Lego and Visualino Distance sensor block were the least appreciated by the children, due to its parameters. They found the distance attributes hard to define and understand its function, although some of them were able to implement the Sensor distance exercise (third exercise).

When children were asked to translate a Visualino program, they had difficulties in doing such task, considering the high number of nodes and branches of the tree. Lego was easier to translate, because it has at most three execution flows easy to translate one-by-one and fewer blocks than Visualino regarding the same behavior.

8.2 Results from the second experiment

The second experiment was separated by two groups, each with its own evaluation goals.

8.2.1 Group 1 experiment

Second experiment's Group 1 had differences in their individual age. Subjects from Team 1 were twelve, Team 2 were ten and Team 3 were eight years old. This experiment started with Visualino language, keeping Lego MindStorms as the second learnt-language.

Learning times

Children from Group 1 had the same learning phase time spent, despite their individual age. The class could only proceed the next learning phase when the subjects had completed the examples tasks. Once more, the learning phase identifies the steep learning curve for each language, through the time spent in each example task. They were able to observe the robot executing the developed programs for each example, understanding the concepts, components and the meaning of the different behaviors.

Tables 8.5 and 8.6 encompass the time spent in the learning phase for Group 1 – the first table describes the first language learnt, being Visualino, and the second table represents the Lego MindStorms case.

Table 8.5: Group 1 time spent for each task example in Visualino

	Sequential Actions (LED + Motors)	Parallel Actions (LED + Motors)	Collision Sensor	Distance Sensor
Time	16 minutes	11 minutes	13 minutes	9 minutes

Table 8.6: Group 1 time spent for each task example in Lego MindStorms

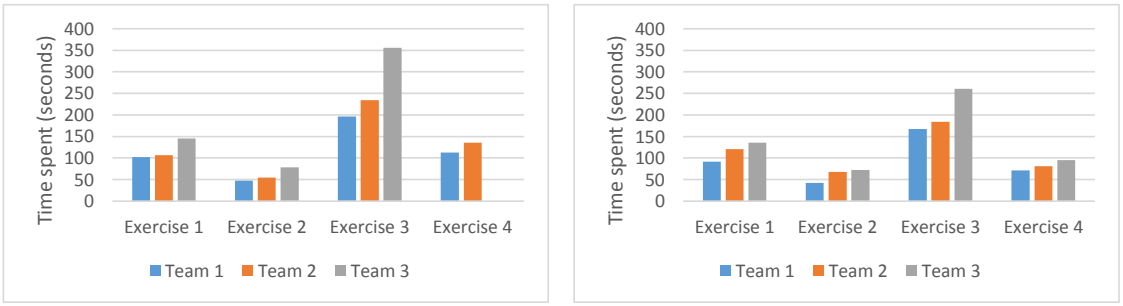
	Sequential Actions (Loudspeaker + Motors)	Parallel Actions with Loop concept	Collision Sensor	Distance Sensor
Time	8 minutes	9 minutes	8 minutes	4 minutes

Visualino learning phase took longer than Lego MindStorms, as the first experiment; children took longer to understand the connections and links between blocks, while in Lego they only needed to drag and drop each block to its corresponding position. However, Lego's learning phase took less than Visualino, considering that the experiment robot's concepts, such as components, sequence of actions, simultaneous tasks and conditions awareness were already taken in Visualino learning classes.

The experiment's evaluator noticed that older children (Teams 1 and 2) learned the language concepts, examples and the robot's components faster than Team 3 subjects (8 years-old).

Visualino and Lego MindStorms results

Once again, it is interesting to evaluate the efficiency of Lego MindStorms and Visualino for each team, considering subjects' age. Efficiency was measured through the time that each team took to complete the designed exercises. Chart 8.11(a) depicts the time elapsed for each team of Group 1, that they took to complete the Visualino exercises. The fourth exercise was not completed by team 3 (8 years-old team), although they implemented the condition from the third exercise. Both teams 1 and 2 took similar times to accomplish the exercise project.



(a) Time spent in Visualino

(b) Time spent in Lego

Figure 8.11: Group 1 time spent in the exercises for each language – started with Visualino

As depicted in chart from figure 8.11(b), the whole group was able to perform the proposed exercise in the Lego MindStorms language. The third and fourth exercises, which encompassed the robot conditions implementation, took longer to complete in Visualino language than in Lego MindStorms. This was expected since children from this group started with Visualino language, having more experience with the robot model itself as language concepts and programming skills.

Satisfaction Level obtained through the Questionnaires

Children's satisfaction regarding each language was obtained through the questionnaires. The comparative questionnaire results are shown in figure 8.12.

Visualino was the most appreciated language in this experiment, specially for Teams 1 (twelve years-old) and 3 (eight years-old) children. Team 1 also answered that none of the languages presented difficulties, while Team 2 subjects (ten years-old) answered that Visualino was less appreciated and harder than Lego MindStorms, justifying that Lego had *an easier way of programming*.

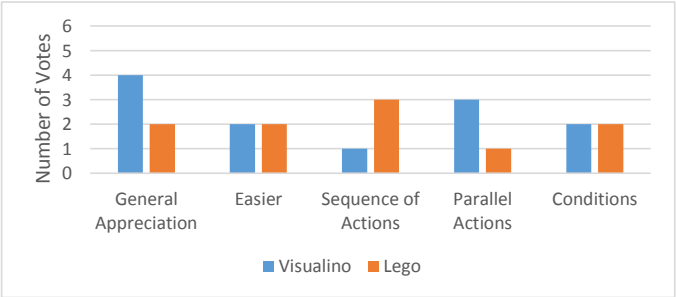


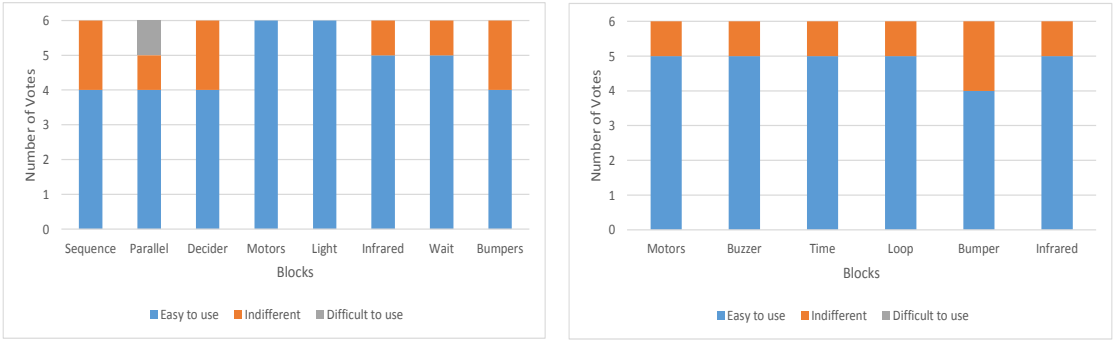
Figure 8.12: Questionnaires answers from Group 1

For the last three questions, Team 1 subjects did not present preference for any language, saying that both languages have the same functionalities through different mechanisms..

When working on a sequence of actions, children from Teams 2 and 3 answered that preferred Lego to develop such behaviors. Children seem to prefer Visualino against Lego when programming *Parallel Actions* on the robot.

Although Team 3 were not able to finish Visualino's final exercise, both subjects answered that they prefer working with Visualino to produce such type of actions. Team 2 showed that they preferred programming conditional behaviors in Lego MindStorms.

Visualino and Lego MindStorms blocks were also evaluated by each team. Graphs shown in figure 8.13 present the children's answers from this experiment. Although Team 1 answered that each block fulfilled their needs, the remaining teams did not expressed the same thoughts. It can be observed that Visualino control blocks presented the higher dissatisfaction level from all blocks. Conditional blocks such as *Bumpers* and *Infrared* sensors, showed that once again, children had difficulties programming conditional tasks.



(a) Visualino's blocks answers

(b) Lego MindStorms' blocks answers

Figure 8.13: Answers regarding blocks ease of use for each Language

8.2.2 Group 2 experiment

ArduinoFlow language was tested with the Group 2 subjects (eight-year old children). Team 3 had already tested Lego MindStorms and Visualino. This experiment has as purpose to test children programming capabilities in a sequential form, as to identify accidental complexities aspects within Eclipse workbench.

Learning Times

The process taken for this experiment has the same features as the previous ones. The learning phase explicits the time spent for each example task presented in slides as the time that children took to understand language and robot's concepts. ArduinoFlow has a limited expressiveness when compared to Visualino and Lego MindStorms – it does not include parallel execution flows, as parallel actions on the robot. Taking this restriction into consideration, the second task from the learning phase cannot be compared to the previous experiments. Table 8.7 shows the time spent for each task example.

Table 8.7: Group 2 time spent for each task example in ArduinoFlow

	Sequential Actions (LED + Motors)	More Actions (Servo positioning)	Collision Sensor	Distance Sensor
Time	15 minutes	4 minutes	10 minutes	6 minutes

The evaluator perceived that children from Team 3 were faster to acquaint the concepts for this new language, although those times could not be measured since the class could only proceed to the next phase, when the subjects had completed and fully understood the example tasks.

Children took longer to understand and actually develop the first task, considering that the first example encompassed the initial contact with the language blocks and the programming actions' paradigm. Once they have overcome the paradigm style, the subsequent phases were relatively easy to complete. They also understood the concepts and

notions behind those examples.

ArduinoFlow exercises results

Group 2 completed ArduinoFlow designed exercise with no errors. The elapsed time for each team is depicted in chart from the figure 8.14.

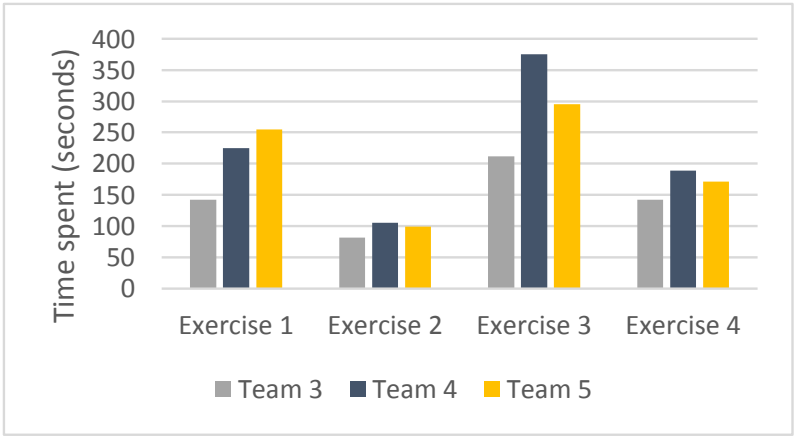


Figure 8.14: Group 2 time spent to complete ArduinoFlow's exercises

Team 3 obtained the best results (for each exercise) in this experiment, thus we may conclude that they developed a better understanding of the robot and the programming concepts it involves, since they learned those from the previous languages; Furthermore, Team 3 as opposed to the other teams, recognized the workbench usability details from ArduinoFlow with Visualino, influencing the results they achieved in a positive manner – the links made between blocks were an easier task for Team 3 than for the others.

The first two exercises – the simple tasks –, regarding a predefined sequence of actions, were easily solved by each team. Although each team achieved successfully the last two exercises, they encountered a visual complexity – the designed model (a big sequence of blocks) occupied too much space for the design surface of the workbench. This issue substantially increased the time spent for both tasks.

Satisfaction Level obtained through the Questionnaire

The chart from figure 8.15 depicts the results from the ArduinoFlow questionnaire. It encompasses the blocks ease of use and its functionalities. Generally, children found ArduinoFlow and its blocks an easy language to program robot behaviors, considering that all of them were able to implement the designed exercise.

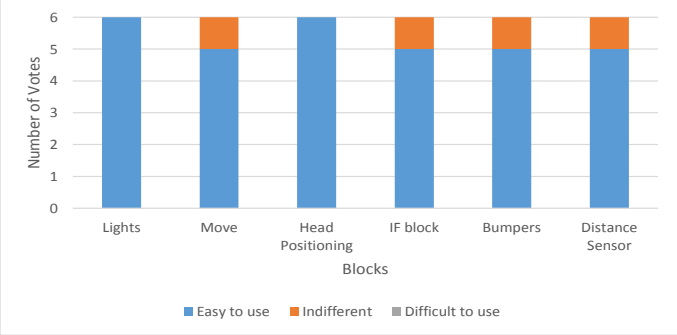


Figure 8.15: Questionnaires answers from both groups

The questions results regarding blocks functionalities, as the language paradigm and design concepts were overall positive.

8.3 Threats to validity

The previous empirical study brought conclusions regarding the usability from the three different languages tested and their respective paradigm. In this section, we show the limitations, possible influences and threats that may comprise the validity of the presented study. We need to take into consideration external factors that are difficult to control during each experiment, as the decisions taken for the global procedure study.

Both experiments took place after lunch, which means that test participants had school activities before the test took place, so possible situations regarding children' weariness, lack of commitment and focus towards the project may be answered due to this factor. We tried to maintain a controlled environment during the activities on both schools, despite their different environments and students.

Concerning the established procedure for the empirical study, we need to determine which aspects may affect the conclusions brought with the analysis. Lego MindStorms language features a perfected and enhanced interface and available functionalities, that may explain the inferior results achieved both for Visualino and ArduinoFlow. Following this reasoning, we were not able to test every functionality and expressiveness from Lego MindStorms that would possible create new lines of thought in children (e.g blocks encapsulation, which means that one block contains several others). Regarding the tests made for each language, we need to advert that the simple tasks (first and second) were easily solved through the learning slides examples – we tried to analyze children skills to create a complex behavior through short and small exercises. During the exercise phase, the participants did not have the opportunity to develop the solution and observe it at real time on the corresponding robot – otherwise, they could detect incorrect implementations easily.

8.4 Conclusion of the analysis

The results presented in the early sections, supplied insights on children mental abstraction and programming capabilities through a diversity of languages with each own paradigm and construction model in the robot domain. We were able to study dataflow paradigm from Lego MindStorms RoboLab, a workflow paradigm from ArduinoFlow programming language and Visualino, which enclosed the behavior tree paradigm.

Through the results for each experiment and tested groups, we are able to determine that children appreciated and achieved better results in the learning phase as the exercise phase with Lego MindStorms VPL than Visualino. Children exhibited superior skills when translating a robot behavior model to their own mental model via the flowchart paradigm (ArduinoFlow and Lego MindStorms VPLs) against the behavior tree paradigm from Visualino.

Although Visualino's exercise did not present such positive results (concerning the time spent and effectiveness) when compared to Lego MindStorms, we are led to believe that the behavior tree paradigm requires a steeper learning curve. We may conclude that the behavior tree paradigm is more appreciated by older children (defined here as ten years old and up), which present greater maturity and cognitive levels. An interesting fact observed in both experiments is that while the schools had small differences regarding their students socioeconomic status, the children who participated in those experiments enjoyed the programming classes and showed similar cognitive and abstraction aspects concerning their programming skills development.

9

Domain-experts language validation

The previous chapters showed the experiment along with its results and conclusions with Visualino's target users. This chapter brings a new experiment taken with Farrusco and Arduino domain experts. The objective is to prove that the language fulfills Farrusco's users needs.

Evaluate Visualino's usability and its learning curve are the main objectives of this experiment. We also tried to obtain informal information from the opinions made by the test participants involved. This can provide information regarding Visualino's negative and positive aspects. It is important to assure a bias-free evaluation through test participants with different expertise levels.

An exhaustive evaluation could be expensive and very time consuming, being difficult to find a time slice of free time from the part of the domain experts. Therefore, this evaluation relies on a comparison of an exercise developed both on Visualino and Arduino languages by the domain experts, and how much time they spent to reach the solution for each language.

9.1 Domain experts profiles and Evaluation Process

The expertise level of the test participants was measured through their working experience with Farrusco and the implementation of behaviors for this specific robot. Thus, we can observe the average time it took each of the domain expert to complete the given exercise (both for Visualino and Arduino). Table 9.1 determines the classification criteria for each test subject.

Table 9.1: Domain experts classification criteria

Profile\Parameter	Experience with Farrusco	Implementation of robot behaviors	Behavior Trees knowledge
Novice-Developer	Medium	Occasionally	None
Advanced-Developer	High	Frequently	Extensive

The test participants were involved in the evaluation process which encompassed three main phases:

- First of all, they need to learn the language organization, interaction method and construction;
- They were asked to perform an exercise test in Visualino. After they completed the mentioned task, they were asked to develop the same exercise in the Arduino textual programming language;
- The final step of this evaluation counted with a questionnaire regarding Visualino's usability.

9.1.1 Learning phase

At this phase, the test participants learned Visualino's concepts through the slides used in the previous evaluation with children. They had the opportunity to test the examples and understand the meaning of each block with its corresponding parameters. The subjects (both novice and advanced developers) took about 20 minutes to practice and understand the given examples and its construction in Visualino.

9.1.2 Exercise contents

The exercise required the test subject to develop an obstacle-awareness behavior, and at the same time Farrusco should change its LED blinking rate according to the situation it confronts. Once again, we are trying to measure the time spent on the exercise implementation, both for Visualino and Arduino. The exercise contained the following list of requirements:

- Farrusco's should move forward at its maximum velocity, and at the same time it blinks its LED every two seconds. Along with this behavior, Farrusco should constantly rotate its Servo.
- If Farrusco detects an obstacle through its Infrared sensor, it should move backwards and turn to a different direction. Simultaneously, the LED's light should be kept lit.
- If one of the bumpers is pressed by any obstacle, Farrusco must alter its light blinking state – now it should blink its LED every second.

  - If the pressed bumper is located at Farrusco's right side, it must move backwards and turn to its opposite direction (left).
  - Conversely, if Farrusco hits an obstacle with its left bumper, it must go back and turn to its right side.

9.1.3 Questionnaire

The questionnaire was composed by two sections: the first one was related to the background of the target user, so we could identify his domain expertise; the second section encompassed several questions regarding Visualino usability, along with open commentaries. The user was asked to rate the questions from the second section, in a scale of 1 (strongly disagree) to 5 (strongly agree). The first section encompassed the following questions:

- Which programming languages are you familliar with?
- How long have you been working with Farrusco?
- Did you ever program behaviors for robots before?
- Could you tell us the level of complexity for those behaviors?

Regarding Visualino usability, the second section of the questionnaire was composed by the following statements, which the test participant should rate according to the scale presented previously:

- I found that the DSL is easy to use;
- I felt very confident while I was using the DSL;
- I found that the language is expressive enough to create a Farrusco's behavior;
- I found that using Visualino I could create complex Farrusco's behaviors more faster than in Arduino C++;
- Visualino appears to be a good programming language for children;
- Rate each one of the following blocks usefulness: Decider Block, Parallel Block, Sequential Block, Motors Block, Time Block, Servo Block, LED Block, Bumpers Block, Distance Block.

Test participants were also asked to expressed their Visualino's opinions through the following inquiry questions:

- If you disagree with any of the previous blocks, tell us why.
- What would you improve in Visualino DSL?
- Which are the positive aspects of Visualino?

9.2 Results analysis

The experiment encompassed two test participants that fulfill the advanced-developer profile – already knew the behavior tree paradigm before Visualino was developed and have been working with Farrusco for more than two years. The other four test participants matched the novice-developer profile – spent substantially less time working with Farrusco and Arduino, less than two academic semesters (1 year). Each test participant was evaluated individually at different sessions. Chart from figure 9.1 show each participant time spent in Visualino and Arduino exercises.

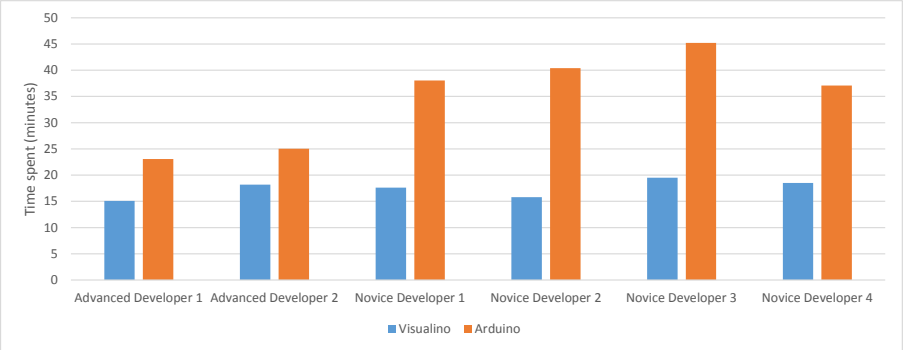


Figure 9.1: Time spent in the exercise for each language and domain expertise profile

As expected, the advanced developers took less time in the Arduino exercise than the other subjects. Nevertheless, all of the test participants took similar times to complete the exercise in Visualino. We can clearly observe the test participants completed the exercise faster in Visualino than in Arduino language.

Figure 9.2 depicts the test participants' appreciation for each block of Visualino. Generally, the overall results appreciation for each block was positive. It is interesting to mention the Decider block was the least appreciated block by the test participants. They explained that the Decider block was the difficult concept and notion to acquire and possibly the hardest to learn for children.

The chart from figure 9.3 shows Visualino's global appreciation, presenting usability aspects such as:

- Ease of use;
- Confidence level when working with Visualino;
- Expressiveness;
- Possibility to program more complex behaviors;
- If it is suitable for children;

The results from this questionnaire's section were overall positive. The advanced developers found the DSL expressive enough (strongly agree) to program both complex

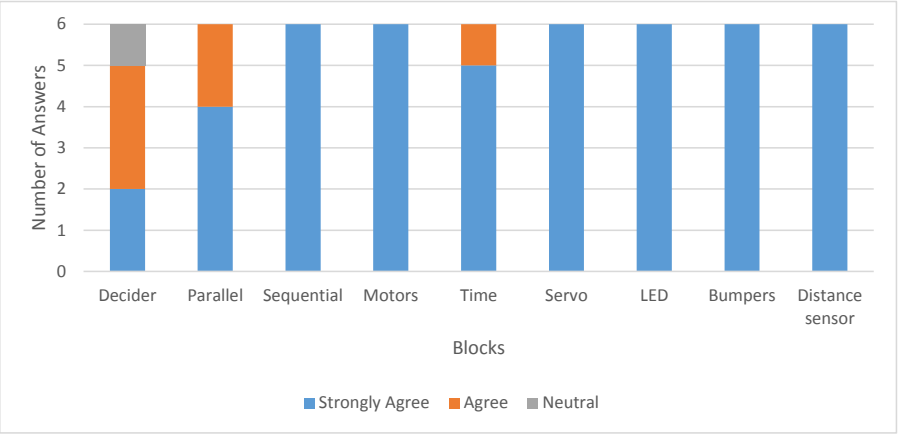


Figure 9.2: Blocks appreciation

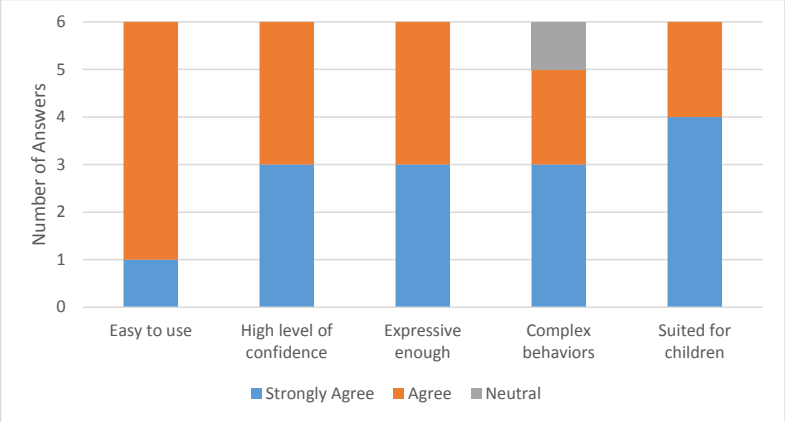


Figure 9.3: Visualino general appreciation

and simple behaviors, while novice developers answered as "agree" considering that they did not know the behavior tree paradigm before.

Participants comments

Most of the test participants agreed that the language concepts such as the blocks and the tree-based paradigm are easy to understand and use, considering the time spent for a robot behavior development and its absence of semantic errors. The test participants also mention that the visual representation of the concepts were also intuitive and clearly reflect the block's purpose.

The decider block was the least appreciated by the domain experts, referring that it was the hardest block to understand and possibly the toughest block that children need to learn.

Test participants suggested some interface enhancements such as:

- Drag-and-drop gesture was missing in Graphical user interface of Visualino;

- The surface design should be bigger so complex behaviors could be developed. This limitation could be solved by creating new leaf nodes of language that could represent modules already programmed;
- The links between blocks could be avoided; for example, when adding a block, it could be automatically connected to its parent node;
- Re-usability of code/blocks by enabling copy-paste functionality.

They – mainly the advanced developers subjects – also proposed the following extra Visualino's functionalities regarding the robot domain:

- Give the user the possibility to set and define different pin numbers for each sensor and actuator;
- Extended sensors and actuators capabilities such as LDR's (light dependent resistor, refered as a light sensor), LEDs actuators with specific colors and more buttons sensors.

9.3 Experiment final remarks

The results obtained through this experiment have shown that Visualino language along with its behavior tree paradigm provides a new approach to developing behaviors for Arduino robots, and specifically Farrusco. Domain experts spent less time when creating behaviors with Visualino than in Arduino language.

Although Visualino contains some usability problems as mentioned previously, the test participants enjoyed its expressiveness and how quickly they were able to develop Farrusco's behaviors when compared with Arduino.

10

Conclusions

In this chapter a summary of the thesis development is presented, along with the main contributions of our work and future work suggestions.

10.1 Dissertation Summary

The objective of this thesis was to develop a DSL adequate for children, so they could program specific robot behaviors. The design process and the implementation of the DSL used Model-Driven Development approach, focusing domain models for the construction of the DSL with its corresponding domain-specific. An empirical study took place so we could evaluate the produced DSL together with similar languages for analogous robots, comparing usability, paradigms, concrete and syntax issues. The studies encompassed the target users, eight to twelve years old children and domain-experts based on their working experience in the robotic area.

10.2 Contributions

One main contribution of this work was to design and develop an adequate DSL for children which targets programming of specific robots behaviors. The DSL is used for education and entertainment purpose by children, in order to aid their development in the areas of Computer Science and Robotics.

We also present an evaluation process instance where we try to validate and identify possible DSL usability problems, based on controlled empirical studies containing exercises, questionnaires and real world observations for a specific children age group. We also compared other languages paradigms so we could determine an adequate language

for each age group. The achieved results also indicate that there is an increased productivity when applying this paradigm model in the robotic domain – although it has its own limitations concerning the user age and different maturity levels when comparing to other visual paradigms. We were able to create the basic design concepts and decisions to the DSL progress and evolution.

10.3 Future Work

Although we believe that this study already supplies important insights of qualitative data regarding robotic programming languages for children, we suggest to perform a large scale study – in terms of population and higher age brackets, since we discovered that the behavior tree paradigm is more adequate to older children. We also suggest to study other language paradigms containing different concrete and syntactic syntax with children, since we were not able to test and evaluate every presented visual languages programming paradigms.

Taking the domain experts comments into consideration, we also suggest that the usability of the DSL should be improved (e.g drag and drop gesture, re-work the visual programming environment). It would also be interesting to proceed with a new evaluation including domain experts from different robot architectures composed by other actuators and sensing components.

Bibliography

[1] E. A. Lee, "Cyber physical systems: design challenges", in *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, ser. ISORC '08, IEEE Computer Society, 2008, pp. 363–369.

[2] D. S. Goldberg, D. Grunwald, C. Lewis, J. A. Feld, and S. Hug, "Engaging computer science in traditional education: the ecsite project", in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, ser. ITiCSE '12, Haifa, Israel: ACM, 2012, pp. 351–356.

[3] A. R. Basawapatna, K. H. Koh, and A. Repenning, "Using scalable game design to teach computer science from middle school to graduate school", in *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, ser. ITiCSE '10, Bilkent, Ankara, Turkey: ACM, 2010, pp. 224–228.

[4] S. Kelly and J. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008.

[5] D. Harel and B. Rumpe, "Modeling languages: syntax, semantics and all that stuff, part i: the basic stuff", Jerusalem, Israel, Tech. Rep., 2000.

[6] E. Marques, V. Balegas, B. F. Barroca, A. Barisic, 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 2012 workshop on Domain-specific modeling*, ser. DSM '12, Tucson, Arizona, USA: ACM, 2012, pp. 13–18.

[7] J. Bézivin, "On the unification power of models.", *Software and System Modeling*, vol. 4, pp. 171–188, 2005.

[8] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography", vol. 35, no. 6, 2000.

[9] A. Rosa, "Designing a DSL solution for the domain of Augmented Reality Software applications Specification", Master's thesis, FCT / Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, 2009.

[10] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley, 2009.

[11] A. Barisic, V. Amaral, M. Goulao, and B. Barroca, "How to reach a usable dsl ? moving toward a systematic evaluation evaluation", Electronic Communication of the European Association of Software Science and Technology, ECEAST, vol. 50, 2011.

[12] N. Bevan, "Extending quality in use to provide a framework for usability measurement", San Diego, CA: Springer-Verlag, 2009, pp. 13–22.

[13] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck, "Taming emf and gmf using model transformation", in *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I*, Oslo, Norway: Springer-Verlag, 2010.

[14] M. M. Burnett and D. W. McIntyre, "Visual programming", *Computer*, 1995.

[15] T. Catarci, M. Costabile, S. Levialdi, and C. Batini, "Visual query systems for databases: a survey", *Journal of Visual Languages and Computing*, 1997.

[16] M. D. Boardman, "Enthusiasm: a tile-based visual programming language for high-level assembly",

[17] D. D. Hils, "Visual languages and computing survey: data flow visual programming languages.", *J. Vis. Lang. Comput.*, vol. 3, no. 1, pp. 69–101, 1992.

[18] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages", *ACM Comput. Surv.*, vol. 36, no. 1, pp. 1–34, 2004.

[19] T. B. Wendy Wenhui Zhang and H. Ye, "Statechart: a visual language for software requirement specification", *International Journal of Machine Learning and Computing*, vol. 2,

[20] X. wen Terry Liu and J. Baltes, "An intuitive and flexible architecture for intelligent mobile robots", in *The Second International Conference on Autonomous Robots and Agents(ICARA), Palmerston North*, 2004.

[21] S. Delmer, "Behavior trees for hierarchical rts ai",

[22] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving behaviour trees for the mario ai competition using grammatical evolution", in *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, ser. EvoApplications'11, Berlin, Heidelberg: Springer-Verlag, 2011.

[23] L. Grunske, K. Winter, and N. Yatapanage, "Defining the abstract syntax of visual languages with advanced graph grammars-a case study based on behavior trees", *J. Vis. Lang. Comput.*, 2008.

[24] S. Papert, *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[25] C. Kelleher and R. Pausch, "Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers", *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, Jun. 2005.

[26] M. Virnes, E. Sutinen, and E. Kärnä-Lin, "How children's individual needs challenge the design of educational robotics", in *Proceedings of the 7th international conference on Interaction design and children*, ser. IDC '08, Chicago, Illinois: ACM, 2008, pp. 274–281.

[27] M. McNally, M. Goldweber, B. Fagin, and F. Klassner, "Do lego mindstorms robots have a future in cs education?", in *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '06, Houston, Texas, USA: ACM, 2006, pp. 61–62.

[28] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: programming for all", *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[29] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: urban youth learning programming with scratch", in *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '08, Portland, OR, USA: ACM, 2008, pp. 367–371.

[30] A. Millner and E. Baafi, "Modkit: blending and extending approachable platforms for creating computer programs and interactive objects", in *Proceedings of the 10th International Conference on Interaction Design and Children*, ser. IDC '11, Ann Arbor, Michigan: ACM, 2011, pp. 250–253.

[31] C. Courage and K. Baxter, *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques*. Morgan Kaufmann, 2005.

[32] U. G. Assembly, "Convention on the rights of the child", in *Convention on the Rights of the Child*, United Nations: UN General Assembly, 1990.

[33] P. Markopoulos, J. C. Read, S. MacFarlane, and J. Hoysniemi, *Evaluating Children's Interactive Products: Principles and Practices for Interaction Designers*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[34] J. Kuper, *International Law Concerning Child Civilians in Armed Conflict*, ser. Clarendon paperbacks. Clarendon Press, 1997.

[35] A. Druin, "The role of children in the design of new technology", *Behaviour and Information Technology*, vol. 21, pp. 1–25, 2002.

[36] A. Bruckman and A. Bandlow, "The human-computer interaction handbook", in, J. A. Jacko and A. Sears, Eds., Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003, ch. Human-computer interaction for kids.

[37] M. Hatzigianni and K. Margetts, "I am very good at computers young childrens computer use and their computer self-esteem", *European Early Childhood Education Research Journal*, vol. 20, no. 1, pp. 3–20, 2012.

[38] D. Chirico, "Building on shifting sand: the impact of computer use on neural and cognitive development", 1997.

[39] K. Precel and D. Mioduer, "The effect of constructing a robot's behavior on young children's conceptions of behaving artifacts and on their theory of mind (tom) and theory of artificial mind (toam)", Children, Youth, Environments Journal.

[40] A. Barišić, P. C. Monteiro, V. Amaral, M. Goulão, and M. P. Monteiro, "Patterns for evaluating usability of domain-specific languages", in *Pattern Languages of Programs Conference 2012*, ACM, Oct. 2012.

Appendix

A.1 Slides used for each experiment

This section contains the slides prepared for each evaluation experiment.

A.1.1 Visualino slides

Visualino slides are presented on the following pages.

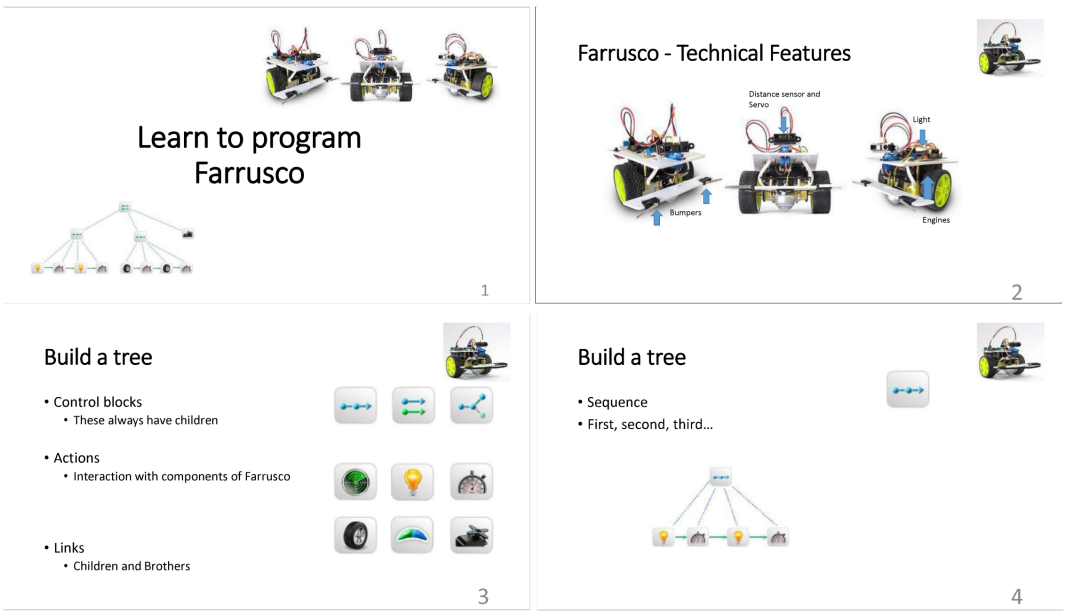


Figure A.1: Visualino's slides – part 1

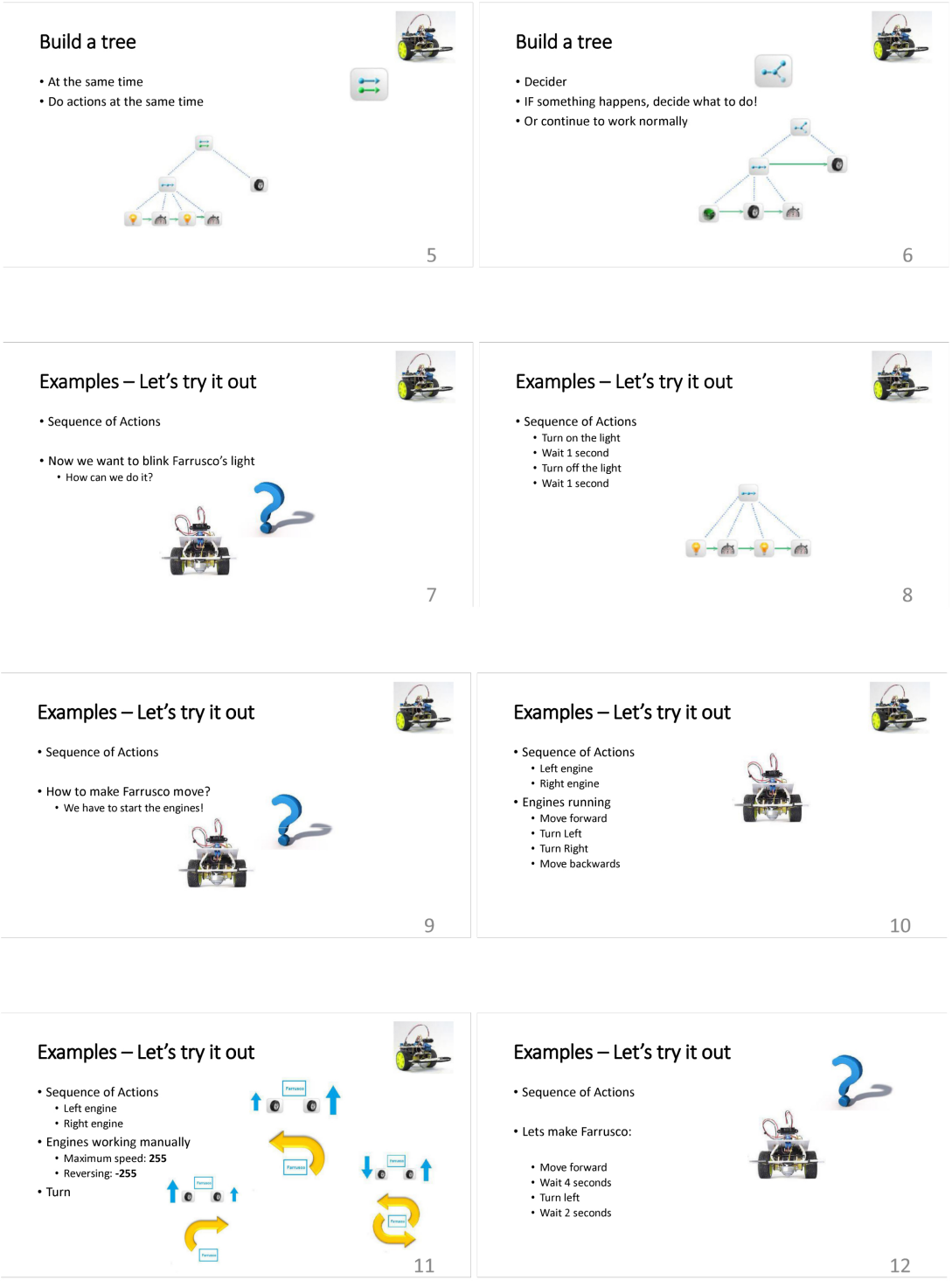
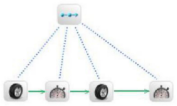


Figure A.2: Visualino's slides – part 2

Examples – Let's try it out

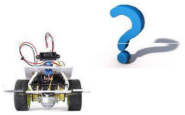
- Sequence of Actions
- Lets make Farrusco:
  - Move forward
  - Wait 4 seconds
  - Turn left
  - Wait 2 seconds



13

Examples – Let's try it out

- Multiple actions at the same time
- How can we make Farrusco blink its LED and move at same time?



14

Examples – Let's try it out

- Multiple actions at the same time
- At the same time:
  - Blinking
  - Move forward

15

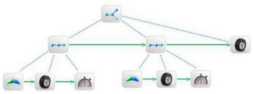
Examples – Let's try it out

- We want a smarter Farrusco
- Move forward
- IF it collides, it should go backwards and turn left for 1 second

16

Examples – Let's try it out

- We want a smarter Farrusco
- Move forward



- IF it collides, it should go backwards and turn left for 1 second

17

Examples – Let's try it out

- Lets make Farrusco even smarter!
- New task:
  - Move Forward

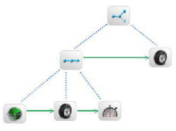
Distance sensor

- IF Farrusco detects an obstacle
  - Move backwards for 1 second

18

Examples – Let's try it out

- Lets make Farrusco even smarter!
- Walk straight ahead
- IF Farrusco detects an obstacle
  - Move backwards for 1 second



19

Small Project

- Imagine that Farrusco doesn't like walls or obstacles
- How to prevent it from getting stuck in a wall or other obstacles?

20

Figure A.3: Visualino's slides – part 3

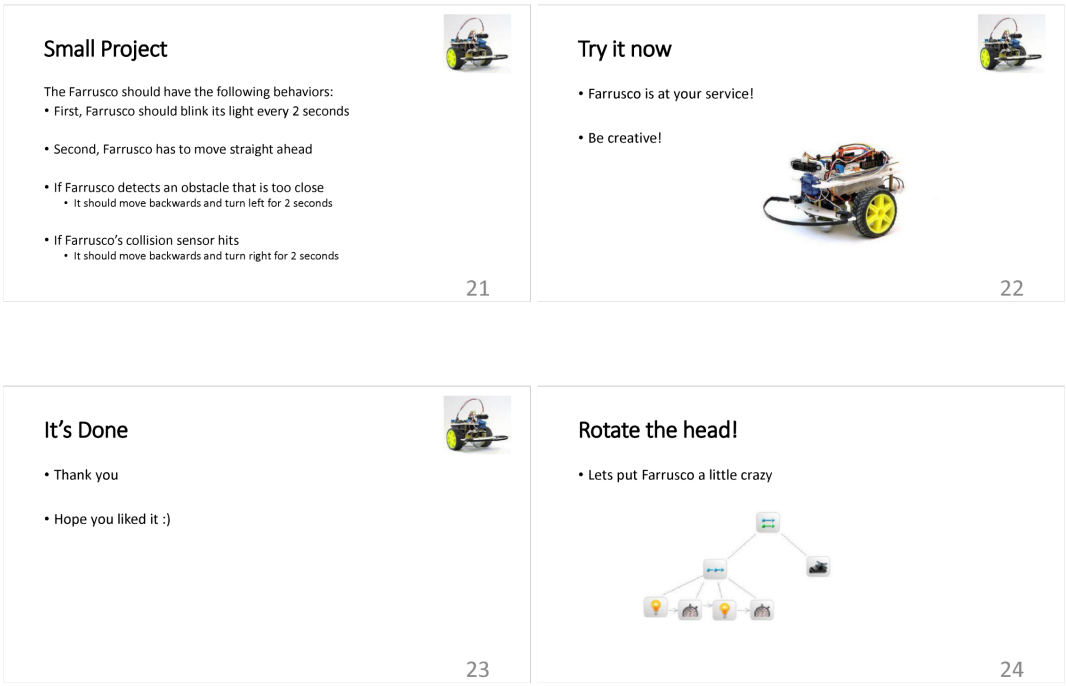


Figure A.4: Visualino's slides – part 4

A.1.2 Lego MindStorms slides

Lego MindStorms slides are presented in this section.

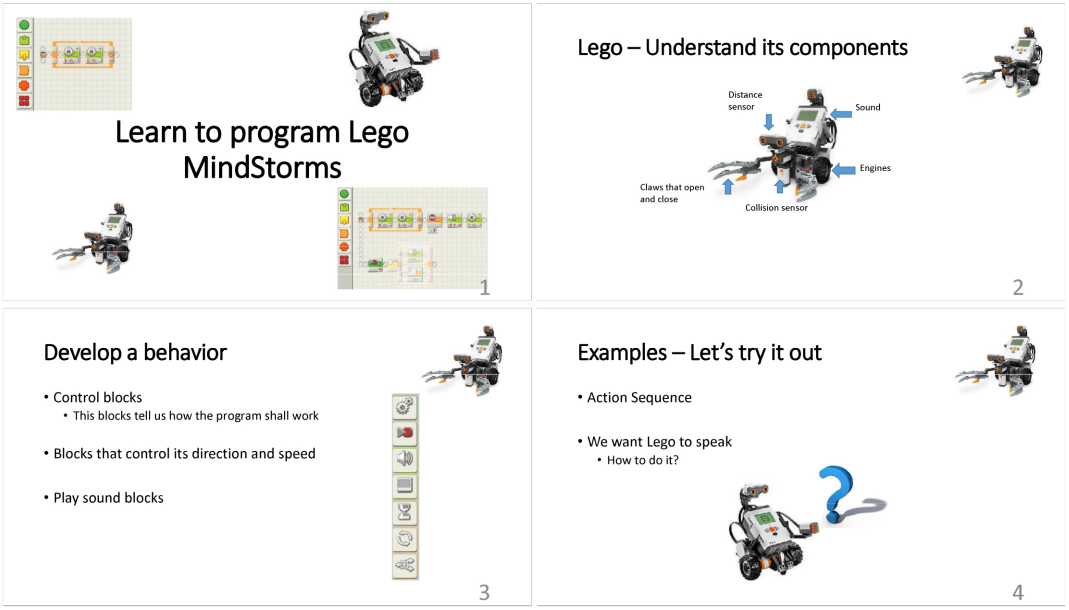


Figure A.5: Lego MindStorms' slides – part 1

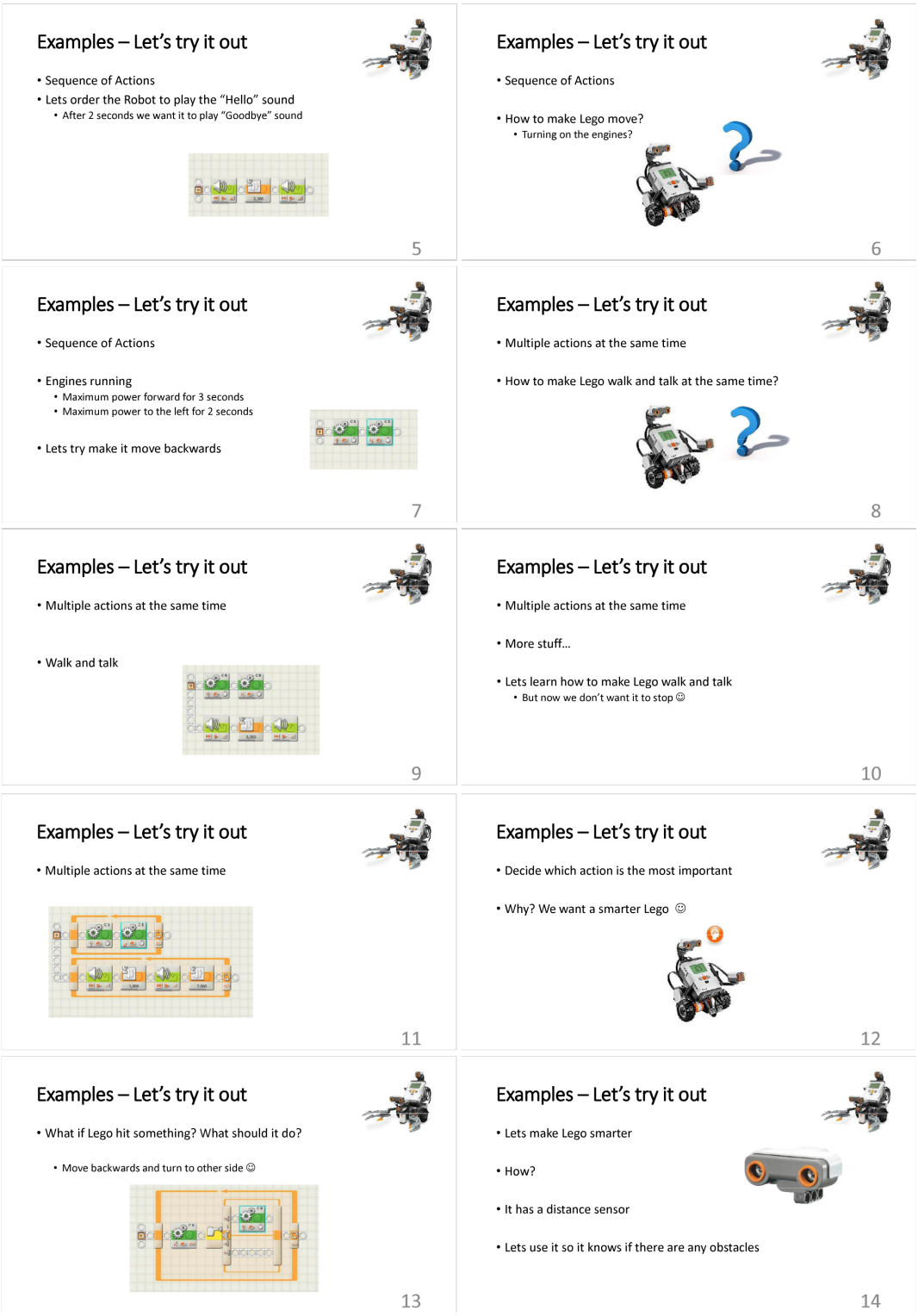


Figure A.6: Lego MindStorms' slides – part 2

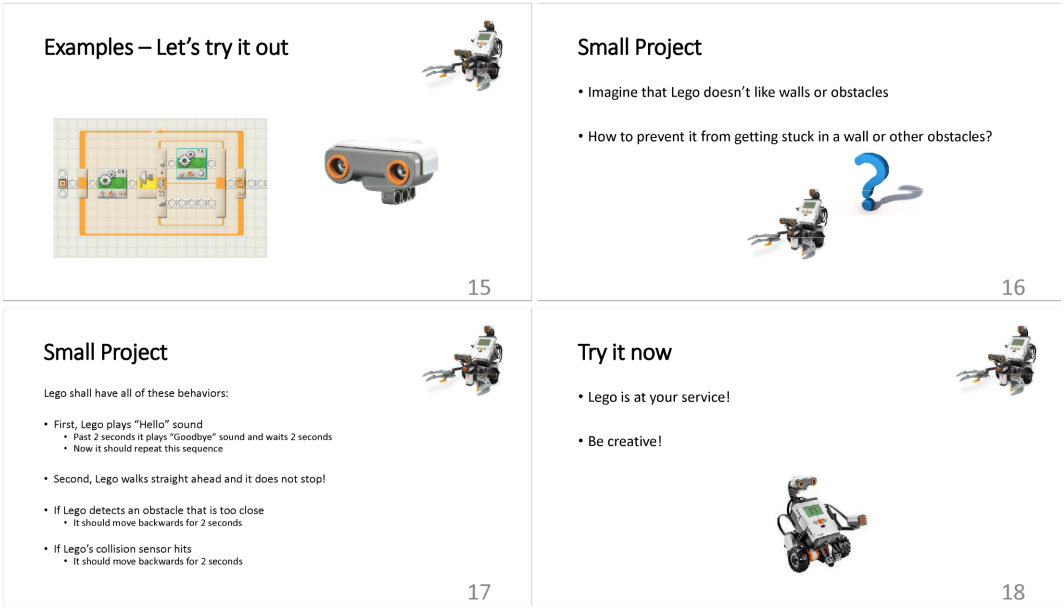


Figure A.7: Lego MindStorms' slides – part 3

A.1.3 ArduinoFlow slides

ArduinoFlow slides are presented in this section.

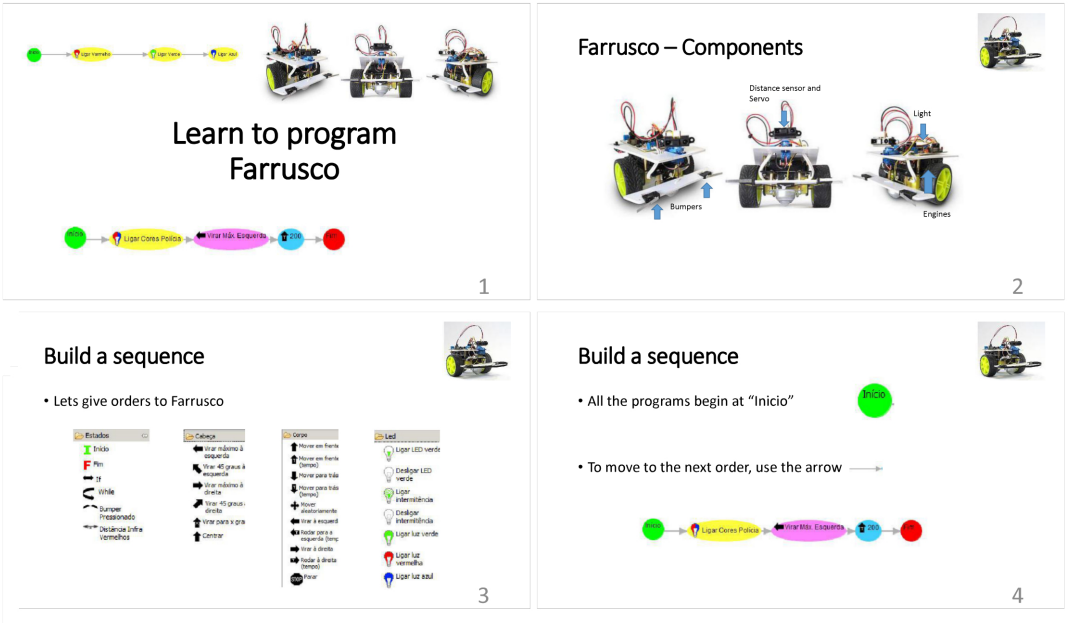


Figure A.8: Visualino's slides – part 1

Build a sequence

- First example,
- Let's change the colors lights from Farrusco

5

Build a sequence

- Second example
- Now, turn ON the engines from Farrusco

6

Build a sequence

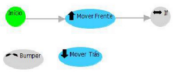
- Third example, orders to Farrusco
- Control the Farrusco's head

7

Time to get Farrusco Smarter

- What **If** Farrusco can observe or feel that it collided with obstacles?
- **If** Farrusco hits, we need to prepare its next action!

. **IF** ...

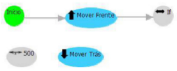


8

Time to get Farrusco Smarter

- What **If** Farrusco can observe or feel that collided with obstacles?
- **If** Farrusco finds an obstacle, we need to prepare its next action!

. **IF** ...



9

Small Project

- Imagine that Farrusco doesn't like walls or obstacles
- How to prevent it from getting stuck in a wall or any other obstacle?

10

Small Project

- First, Farrusco has to turn on the light
  - First green, for 2 seconds
  - Then red, for another 2 seconds
  - It must repeat these actions
- Second, Farrusco must move forward for 5 seconds
- If Farrusco detect an obstacle that is too close
  - It should move backwards for 1 second
- If Farrusco's collision sensor hits
  - It should move backwards for 1 second

11

Try it now

- Farrusco is at your service!
- Be creative!

12

It's Done

- Thank you
- I hope you liked it :)

13

Figure A.9: ArduinoFlow's slides – part 1