David Monteiro Duarte

Licenciado em Ciência e Engenharia Informática

A Model-Driven Approach for Mobile Business Intelligence

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientadores : Prof. Doutor Vasco Miguel Moreira Amaral, Professor Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Nuno Pereira Gomes, Geographical Information Systems Consultant, Novabase

Presidente:	Doutor João Miguel da Costa Magalhães
Arguente:	Doutor Ademar Manuel Teixeira de Aguiar
Vogal:	Doutor Vasco Miguel Moreira do Amaral

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2016

A Model-Driven Approach for Mobile Business Intelligence

To my mother, Francisca

Acknowledgements

I would like to express my sincere gratitude to everyone who directly or indirectly supported me throughout these last academic years.

First I would like to thank the teacher Vasco Amaral, for his availability to collaborate with me on this dissertation, for all the advice and knowledge shared. Your help was indispensable.

To my college Faculdade de Ciências e Tecnologias from Universidade Nova de Lisboa, in particular to Computer Science Department for all the knowledge and opportunities transmitted during the last five years.

A special thanks to Novabase enterprise group, for the conceived opportunity, trust and the knowledge shared. In particular, to Nuno Pereira Gomes for accepting to collaborate with me, and to all the colleagues I have met on this company, for all the moments shared with me, it made easier this chapter of my academical life.

To everyone who accepted to spent some of their time to participate on the usability tests. Your contribution was crucial to the realization of this work.

To my family, specially to my parents, a paragraph is not enough to thank you for the constant love, support and values taught, which contributed, in all aspects, to my personal growth.

To everyone whose name I am not mentioning here, but contributed directly or indirectly to the realization of this dissertation.

To everyone, thank you!

Abstract

The concept of Mobile Business Intelligence is nowadays gaining prominence in business markets. With the emergence and evolution of mobile technologies such as smartphones and tablets, the users gain the opportunity to analyze the corporate information, anywhere and anytime, based on charts, tables and dashboards. However, there is also the question of how to provide the freedom to the user to build its own analytical components.

This work will address the problem of developing a hybrid mobile solution towards the Business Intelligence domain, offering monitoring services and simultaneously addressing the problem of user empowerment, with easy configuration and semi-automatic generation of analytical widgets. To provide such capacity to the user, the proposed solution is based on the design of a Domain Specific Modeling Language, aligned with the Model-Driven Development approach and inspired by the Product Lines principles.

The last part of this work is dedicated to evaluate the language usability based on an empirical test, executed by a set of subjects with different backgrounds of specialization. In this sense, we define two groups: end users and domain experts. The goal is to determine the extent to which the prototype can be used to empower the end users. As support for the analysis we have extracted a set of measures, alongside with the final appreciation from the domain experts group, composed by people currently working on Business Intelligence.

Keywords: Model-Driven Development, Business Intelligence , Mobile Technology

Resumo

O conceito de *Mobile Business Intelligence* encontra-se em destaque no mercado empresarial. Com o aparecimento e constante evolução de tecnologias móveis tais como *smart phones* e *tablets*, os utilizadores ganham a oportunidade de analisar a informação de negócio em qualquer altura e em qualquer lugar, sob forma de gráficos, componentes tabulares e painéis interativos. No entanto, surge também a questão de como atribuir liberdade ao utilizador de modo a construir os seus próprios componentes analíticos, configurando a sua própria visão sobre os indicadores de negócio.

Com este trabalho pretende-se desenvolver uma solução móvel híbrida de *Business Intelligence*, onde essencialmente se pretende dotar os utilizadores de capacidades de monitorização e paralelamente endereçando o problema de *user empowerment*, da configuração de componentes analíticos. Nesse sentido, a solução proposta passa pelo desenho de uma Linguagem de Modelação Dominio Específico, implementada usando abordagens de Desenvolvimento Orientado a Modelos e inspirado nos princípios de Linhas de Produto.

A última parte deste trabalho é dedicado à avaliação da usabilidade da linguagem, baseando num teste empírico com a participação de um conjunto de utilizadores com diversos percursos formativos. Nesse sentido, foi definido dois grupos: utilizadores finais e especialistas do domínio. O objetivo é determinar até que ponto o protótipo desenvolvido é capaz de capacitar os utilizadores finais na configuração dos seus componentes. De modo a suportar as análises, extraímos um conjunto de métricas, juntamente com a apreciação final de profissionais do domínio, sendo indivíduos que atualmente trabalham na área de *Business Intelligence*.

Palavras-chave: Desenvolvimento orientado a Modelos, *Business Intelligence*, Tecnologia Móvel

Contents

List of Figures

List of Tables

Listings

1

Introduction

The ever-increasing innovation in mobile technologies is leading to more services being offered to end users, either for personal or professional purposes. It is no novelty that the organizations recognize this trend and the opportunities in their business activities, namely the possibility of empowering customers, partners and employees, wherever the location and the context of the moment [SM03].

Business Intelligence (BI) is no exception [MPM15]. Converging BI and mobility, results in the opportunity to manipulate corporate data and deliver useful knowledge anywhere and anytime, leveraging the productivity of domain users using ubiquitous applications that provide a presentation of business related information elements as key performance indicators (KPIs), dashboards or reports.

As users discover insights, new business related questions arise. Explaining why Ryan Mcshane [MPM15] argues that Mobile Business Intelligence, should not only provide static visual data but also let users dive deeper, asking and answering questions by means of visualization, filtering and drilling. Additionally, the same author points out that, in some cases, there is a need of better understanding regarding the underlying data, for instance, "can you add order data to the sales report to answer an entirely new question?". In this sense, as new requirements appears, new visual components are necessary. However, the development of such components requires a full-stack of technical knowledge, from front-end designing to back-end data-modeling. Requiring such knowledge can affect negatively the end user productivity because they are chained to IT resources to extend the mobile solutions, requiring considerable time and effort to produce tangible results. This is a concern that gains prominence in traditional BI systems wherein business users are fully dependent on technical staff to gather their information needs through standardized reports and dashboards.

The solution resides in adjusting the workload between the parts, by conceiving to the end users the ability to construct their analysis, on demand and regarding their needs. However, integrating ubiquitous applications alongside with user-empowerment features in BI architectures poses some challenges, as described in Table 1.1: Mobile development costs; The availability of data; Mobile resources limitations; User empowerment; Corporate information security.

Table 1.1: Challenges description - Adapted from [Cha+13]

Challenge	*Description*
Mobile development cost	The vast diversity of existing devices makes the development expensive in terms of time and resources costs. For instance, how to construct an application compatible with multi-platforms?
Data availability	It is important that mobile applications deliver available and up-to-date information, regardless of possible offline status and locations.
Dealing with mobile resources limitations	Resources of mobile devices typically are more limited in comparison, for example, with desktop computers. Storing, processing and rendering can be a problem if the device cannot cope with the volume of the data and dimensions of the visual components.
User empowerment	Self-service solutions bring challenges as end users are not required to have the technical knowledge to make proper use. Thus this tools must abstract the complexities as well as offering expressiveness focused on the problem space of the domain.
Corporate information security	Data mobility and user empowerment brings security concerns related to the business information, as unauthorized users could easily access and explore the corporate data.

1.1 Problem Statement and Motivation

Given this set of challenges, we will focus on the user empowerment issue in the context of analytical components construction, for mobile devices. As aforementioned, the development of mobile applications requires considerable amount of effort and a background of knowledge that requires technical expertise, affecting the end users productivity.

For instance, Novabase [1], an IT organization, based on the current technology trends and aligned with their organizational opportunities, has currently developed a Mobile BI solution for the energy sector, targeted to business users for monitoring purposes,

[1] http://www.novabase.pt/pt

through dashboard visualization on tablet devices. The motivation was to have a customizable and extensible solution where they have full control regarding which components are delivered to the clients. However, the analytical content delivered is predefined and strictly coupled with the business concepts, which means that, the current prototype does not provide any means of user customization and self-construction of analytical components. The user empowerment is an important characteristic because they, better than anyone, know what information they want to visualize. To cope with the constant user needs, the current architecture would require the allocation of IT resources on maintenance tasks that would be better allocated on horizontal improvements of the application. Also, as a consequence of developing pre-defined analytical components, if new clients arise, the contents will need to be rebuilt, namely the data queries and the data mapping on the visual properties. By conceiving a strategy, on which the business users are empowered with the ability of configuring generic components, these challenges will be covered.

The current state-of-the-art related to the topic of business user empowerment, on BI domain, is quite vast. Most of the existing technologies provide several approaches to tackle all the challenges previously mentioned. Despite the capability to provide extensive and powerful analytical features, some of the tools follows a "One size fits all" approach, in the sense that they provide a means of data analysis that fits all type of users, from regular to advanced users. This is why, the author on [Eck13] argues, based on statistical results from a survey made on 2012, that this tools are difficult to use for business users.

Similarly, Imhoff counsels BI managers with an example: to provide a buffet of BI components. The goal is to BI teams construct a starter library of contents as part of a self-service BI system, so business users can pick and choose what they need based on their requirements.

At this point we can formulate our research question in the following way: Is it possible to empower the users so that we can establish a mean of cooperation between the business and IT groups?

1.2 Objectives

The objective is to define a way to solve the business user empowerment issue in a context of mobility. It will be analyzed an approach of cooperation between IT and business users, to provide a level of freedom that enables the latter to build analytical components, based on the customization of templates (assets). The final goal is to deploy and integrate those components on a host mobile application optimal for tablet devices.

We will direct our approach towards the Model-Driven Development abstraction and

(semi) automation techniques to provide the business users with the capability of customizing and generating the components mentioned above with model-to-text transformations, given the end user specification in a concrete syntax of a Domain Specific Modelling Language (DSML) that abstracts from the technical concepts, e.g. from general purpose languages.

Intuitively, and derived from the Software Product Line concepts, the IT staff (BI consultants), which will be considered as domain experts throughout this document, will play the role of product line developer, to define the assets. The business users, which will be considered as the DSML end user, will consume those assets in order to define the variable aspects, by customization of the analytical widget template.

In fact, the Model-Driven Development concept will be the core of the solution as it will enable the end users, who are not required to have technical skills, to develop their analytical components thanks to the key concept of model-to-text transformation that will generate the code from the model specification. On top of that, a vast quantity of traditional model manipulations will be available to cover other specific goals. As such, this strategy, besides covering the research question, also brings the opportunity for further research and development to address additional uses cases with a model based solution.

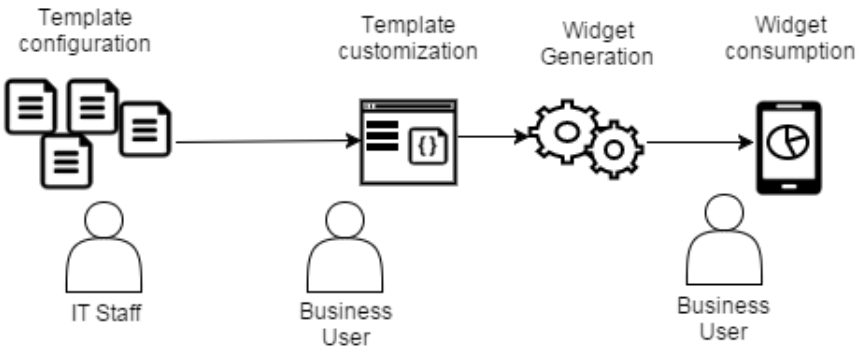


Figure 1.1: Generic architecture

1.3 Context

This work will be developed in the context of a project at Novabase, an IT company specialized in software development. Historically, this company was born in 1989 as a start-up. During the second half of 1990, has positioned as an umbrella organization, constituted by a network of enterprises.

Novabase vision is to simplify the quotidian of people and companies, based on Information Technology, acting on several sectors: Telecoms&Media, Financial Services, Government, Healthcare&Transports, Energy&Utilities.

In particular, our work will focus on Energy&Utilities sector, in the Energy Core Solutions team (ECS). The ECS team is vast and has a strong background of scientific, technological and business knowledge, resultant from several years of experience on constructing solutions for this sector.

1.4 Contributions

As primary contributions, it is expected the following:

- Development of a prototype for BI whose core will be aligned with the concepts of Model-Driven Development, offering a Domain Specific Modelling Language that will serve as the support for end user customization and inspired by the concepts of Software Product Line;
- Conception of a prototype solving the user empowerment issue, and aligned with the portfolio of solutions of the department;
- Review and extension of the current mobile prototype implemented by the ECS department;
- Conception of an usability evaluation whose lessons learned will also serve as input and baseline for future work.

1.5 Document Structure

The dissertation is structured on the following way:

- The chapter 1 introduces the reader on the theme of this thesis, presents the problem statement and delineates the objectives and contributions of this work;
- In chapter 2 is provided a theoretical and technical background of concepts related to this dissertation;
- In the chapter 3 is presented the related work concerning the user empowerment issue in the Business Intelligence area;
- The chapter 4 introduces a case study example which will be used throughout the dissertation. Additionally, we took advantage of the case study to present the overall solution, which describes, in a generic perspective, the modules needed to conceive the prototype and the procedure for its construction. Lastly, it is presented the prototype front-end with a running example;
- At this point the reader should have a generic understanding regarding the solution. As such, in the chapter 5 we will start to detail the prototype conception by explaining the modeling language concepts, including the abstract and concrete syntax;

- In the chapter 6 the implementation is described. Presenting the development, extension and integration of each architectural module;
- In the chapter 7 it is presented the validation process, where it starts with the description of the evaluation setup. The chapter ends with the analysis of the extracted results - including the validation of domain experts;
- In chapter 8 we present the conclusions and future work.

2

Background Concepts

In this chapter we cover theoretical and technical background concepts related with this dissertation. Foremost, we explain some theoretical concepts that align with the user empowerment issue, then we perform a domain analysis by presenting theoretical concepts related with the Business Intelligence area. In the technical background the presentation of the first Mobile BI prototype developed by Novabase takes place, along with other relevant aspects like Modeling Frameworks used on the implementation of Domain-Specific Languages.

2.1 Theoretical Background

2.1.1 Model

Thomas Kuhne defines a model as "an abstraction of a system allowing predictions or inferences to be made" [Küh06]. This definition covers the generic idea of a model, as being a conceptual process that generalizes and describes a target object by capturing its essential properties [Küh06]. A model, must then, be capable of reducing unnecessary information, rather than just being a full copy of the original system represented.

A meta-model, on the other hand, is a model of another model [Küh06], just like a meta-data is data that describes other data[1]. Kuhne's explains in [Küh06], that one can regard a meta-model as artifacts that defines the vocabulary and constraints used in models. Therefore, if a model is a conceptual process of a system, a meta-model is yet another abstraction that captures the general properties of one or several instances of models.

[1] http://www.oxforddictionaries.com/definition/english/metadata

2.1.2 Domain-Specific Language

Concerning to the definition of domain, dictionaries[2] generalizes it as "an area of knowledge or activity". Thus, a domain can be a business area, for instance, banking, healthcare, energy. But also a technical domain, whose concepts are horizontal to different business areas. The authors on [KT08] defines those as vertical and horizontal domains, respectively.

A Domain-Specific Language (DSL) is then, a computer programming language that is oriented to a specific area of knowledge or activity, it concerns to a particular range of problems of a particular domain [Fow10].

Thus, it raises the level of abstraction in comparison with general purpose languages (GPLs) [3], by addressing the specification directly to the concepts of the problem space. It sacrifices generality and flexibility in order to leverage simplicity, quality, the minimization of the accidental complexity and the increasing of the end user productivity.

Another important characteristic is the representation format readable by the end-users, also denominated as concrete syntax [Gre+04]. A DSL can have textual or graphical notation, that, in common, conforms to elements and rules defined on a abstract syntax.

2.1.2.1 Domain-Specific Modeling Language

Modeling languages provides the syntax and semantics to models. Domain-Specific Modeling Languages (DSML) limits the modeling specification (either textual or visual) to the concepts and properties of the domain.

2.1.2.2 Abstract Syntax

The abstract syntax defines the individual elements of a language and the rules that defines how those elements can be combined [Gre+04].

The abstract syntax is, therefore, a type of meta-model structure that captures, in a abstract form, the domain properties of the DSL.

With the definition of the abstract syntax it is possible to define several instances of textual and graphical concrete syntaxes that are scoped to the concepts related with the domain. This is useful when the development of the DSL is concerned with the involvement of the domain users, as a list of options, towards the more suitable concrete syntax.

2.1.2.3 Concrete Syntax

The concrete syntax is the format of the language, is what the user interacts with [Voe+13], the general categories are textual or graphical. Textual syntax tend to be more flexible in terms of detailed formalizations, like algorithms. Graphical notations, on the other hand, are better in describing concepts like relations or flow [AH10]. However, the choice is not

[2] http://www.oxfordlearnersdictionaries.com/definition/english/domain
[3] Examples of General Purpose Languages: C, C++, Java, JavaScript

mutually exclusive, i.e, a language can be represented via textual and graphical format, synchronized with each other, to present different levels of formal specifications.

2.1.2.4 Transformations

The author on [Gre+04] defines transformation as "a process that creates or modifies one or more output specifications, from one or more input specifications". Thus, given a model expressed with a modelling language, for instance with a DSML, a vertical transformation process can be used to generate a lower abstraction level of specification. This introduces the notion of Model Driven Development, as methodology of development based on models [Gre+04].

2.1.3 Model-Driven Development

The author on [Voe+13], refines the definition of model by pointing out two types of models, descriptive and prescriptive. A descriptive model is a representation of an existing system, that can be used for requirement analysis and documentation purposes [Voe+13]. On the other hand, a prescriptive model is used to (semi)automatically construct the target system [Voe+13]. This characteristics introduces the concepts of Model Based Engineering (MBE) and Model-Driven Development (MDD), respectively.

MBE paradigm, is the case where descriptive models play an important role on the life-cycle of software development. However, these artifacts do not drive the development process, as they do not generate the implementation. The development is rather code-driven, in the sense that, developers read the model and manually code the software, this is a tedious process and error-prone.

MDD solves this problem basing upon prescriptive models [Voe+13]. Rather than just representation artifacts, they raise the level of abstraction and drive the development process, based on model-to-text transformations to generate the implementation. Therefore, increasing user productivity and software quality.

Figure 2.1: On the left side a depiction of Model-Based Engineering, on the right side a depiction of Model-Driven Development

2.1.4 Software Product Line

MDD is the driver for a methodology that provides abstraction and automation techniques based on the specification of prescriptive models that generates a implementation. Nonetheless, and according to [GS03], it focus only on the generation of a single product at a time, when most of applications are member of families that share common sub-problems. For that reason, a Software Product Line (SPL) concept appears as a production capacity for a set of software products that shares common and variable set of features, built from a common set of production assets that captures the knowledge about how to produce the family members[Gre+04].

As represented on 2.2, a product line development uses two distinct but collaborative development processes:

- **Product line development:** Defines the kinds of products developed (the product line scope), builds and supplies the production assets for the product development [Gre+04];
- **Product development:** Applies the aforementioned assets to produce products, and provides feedback to the upstream product line development for continuously improvement of production assets [Gre+04].

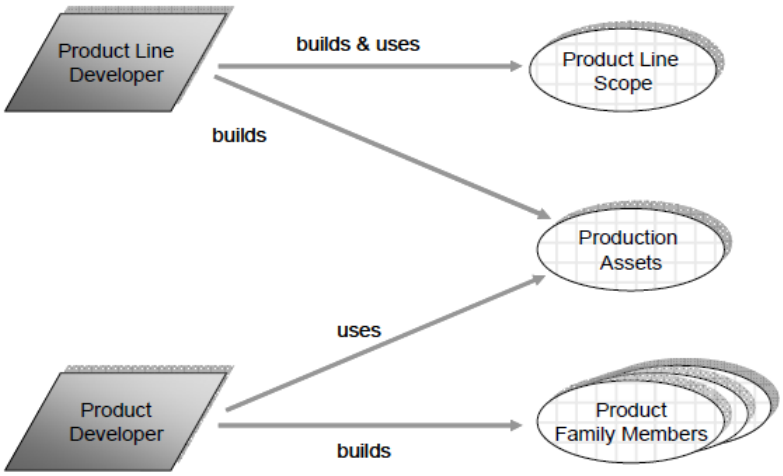


Figure 2.2: Overview of a Software Product Line [GS03]

2.1.5 The Domain of Business Intelligence

The domain whose concepts are being addressed to develop the solution for the user empowerment issue is related with Business Intelligence (BI), which is an area broadly recognized by organizations as the support for decision making activities. It is, basically

a domain, wherein its concepts are horizontal to diverse business areas. Hence, it is fundamental to understand this domain. As such, in this section we provide an explanation of concepts related to BI domain, which will be the support and the scope for the entire cycle of development.

The sources of domain knowledge were acquired from the literature of Business Intelligence and Data Warehousing area alongside with the collaboration of experts on the area.

2.1.5.1 Historical perspective of Business Intelligence

Historically, Business Intelligence (BI) was first defined by H.P Luhn in 1958. In his paper, he describes an automatic system, denominated as Business Intelligence System, to distribute information to different organizational sections accordingly to their interests and needs. He starts by defining the term Business as a "collection of activities carried on for whatever purpose" and Intelligence as the "ability to apprehend the interrelationships of the presented facts in such a way as to guide action to a desired goal" [Luh58].

Later on, as new approaches to store data appeared (relational model by Edgar Codd in 1970) and easier ways to access and analyze information were provided by the emergence of data-warehouses and new set of tools offering decision support functionality, Howard Dresner based on Luhn's ideas, refined the BI definition as an umbrella term that "includes the applications, infrastructure and tools, and best practises that enable access to and analysis of information to improve and optimize decisions and performance" [Ele11].

2.1.5.2 Business Intelligence Generic Architecture

Longbing Cao et al. on [CZL06], defines the process of constructing a BI system by extracting, transforming and loading data stored in disperse enterprise information systems (EIS), into a data warehouse (DW) for linkage with a visualization area. These components forms the generic BI/DW environment [KR02]

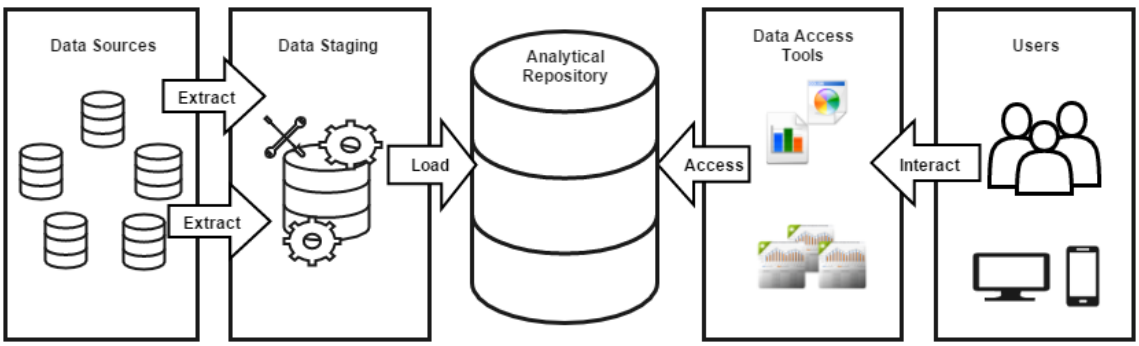


Figure 2.3: Business Intelligence generic architecture adapted from [KR02]

- **Source Layer**: These are the enterprise information systems which stores operational data (Data Sources). The common goal of the next layers is to refine this data and extract useful knowledge from it;

- **Extract Transform and Load Processes**: Extracting, transforming and loading (ETL) data from disperse sources into a data warehouse (DW) [CZL06]. As an intermediate step, there is a staging phase (Data Staging), where the extracted data is dumped for "cleansing, combining data from multiple sources, de-duplicating data, assigning warehouse keys" [KR02];

- **Analytical Repository**: This is where the information is loaded to. It comprises an historical warehouse of data (DW), where the latter is organized, stored and available for data-driven support systems;

- **Data access tools**: This layer comprises the tools that access the stored data and supports the decision making processes of the users. It comprises tools like predefined and/or Ad-Hoc reporting, dashboards presentation, data-mining engines. [CZL06]

2.1.5.3 Data Warehouse implementation

The data warehouse objective is to provide a centralized, consistent, easily accessible and understandable source of historical information.[KR02]

Regarding the implementation, two paradigms arises: Immnon and Kimball approach. Both agree on having a central repository of information for decision support systems, the difference resides in how to design it.

The Immnon approach follows a top-down pattern. His proposal is to model a central enterprise data warehouse in a third normal form (3NF). Only then, the definition of subjected oriented, multi-dimensional and standalone data repositories takes place. These are defined as data-marts.

On the other hand, Kimball defends that normalization techniques used for designing operational systems should not be used to model data warehouses [KR02]. His proposal is to define a bottom-up approach, by designing multi-dimensional data-models that are linked together, resulting in a centralized enterprise and denormalized data warehouse.

Regardless the approach used, data modelling is an important milestone on the construction of data warehouses. It describes the properties and relations between entities, defines the structures that stores the information.

In analytical scenarios, the technique most used is **multi-dimensional modelling** [KR02], it packages the data in a format that leverages understandability, query performance, and resilience to change [KR02].

2.1.5.4 Conceptual Multi-Dimensional Model

From a conceptual view, a multi-dimensional model can be represented as a cube, as represented in the figure 2.4, or hyper-cube for more than three dimensions. The cells of the conceptual cube are the thinner elements, these represents the business measurable attributes that are bounded by the dimensions. The latter represents the edges of the cube, and give contextual information by providing multiple points of view and aggregation levels over the business related events. For example, sales amount of a specific product in a particular city by year, with the capability of drilling down to the month level.

The navigational features are possible with the definition of hierarchies. These structures combines the dimension elements with parent-child relationships. It enables the analysis at different levels of aggregation. For instance, drilling down from sales amount by year to month or rolling up from week to month level, where the relationship $Year \leftrightarrow Month \leftrightarrow Week$ forms a hierarchy.

Operations through the cube: The cube and the hierarchies provides operations that leverages the analysis. Common operations are:

- **Slice:** It is, essentially, returning a square subset from the cube, by choosing a single value from the chosen dimension and retrieving the cells contained in it. For instance, returning sales amount from the year 2015 - figure 2.5;
- **Dice:** It produces a cubic subset from the main cube, by picking multiple values from multiple dimensions - figure 2.6;
- **Drill down:** It steps down the concept of hierarchy, aggregating the measures from higher and summarized to lower and detailed levels;
- **Roll/drill up:** It is the reverse of drilling down. Navigates from lower to higher and summarized levels;
- **Drill across:** Join measures from different cubes, across conformed dimensions, i.e., the ones which are shared by the measures. For instance, joining two measures from different cubes, both related with time dimension, in order to calculate a ratio, by year.

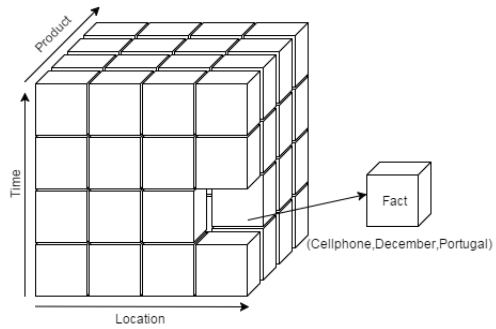


Figure 2.4: Conceptual cube

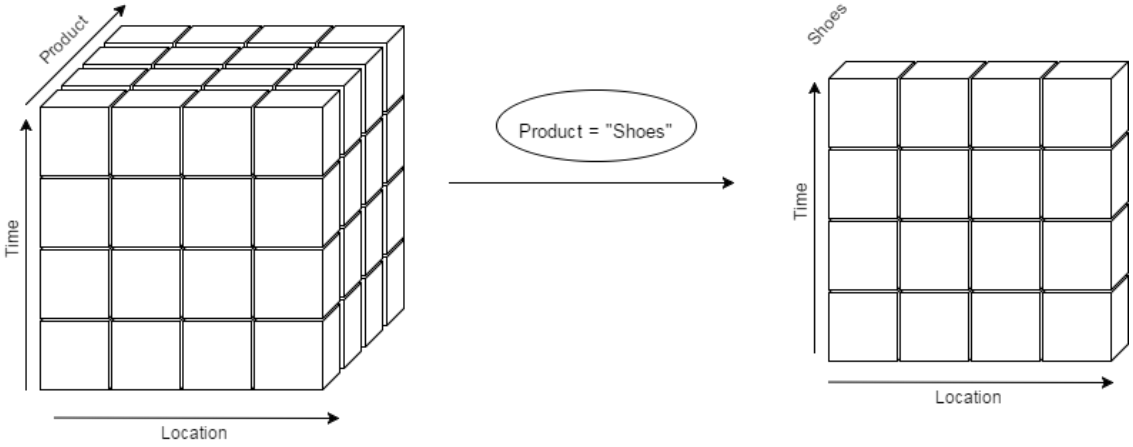


Figure 2.5: Cube slicing

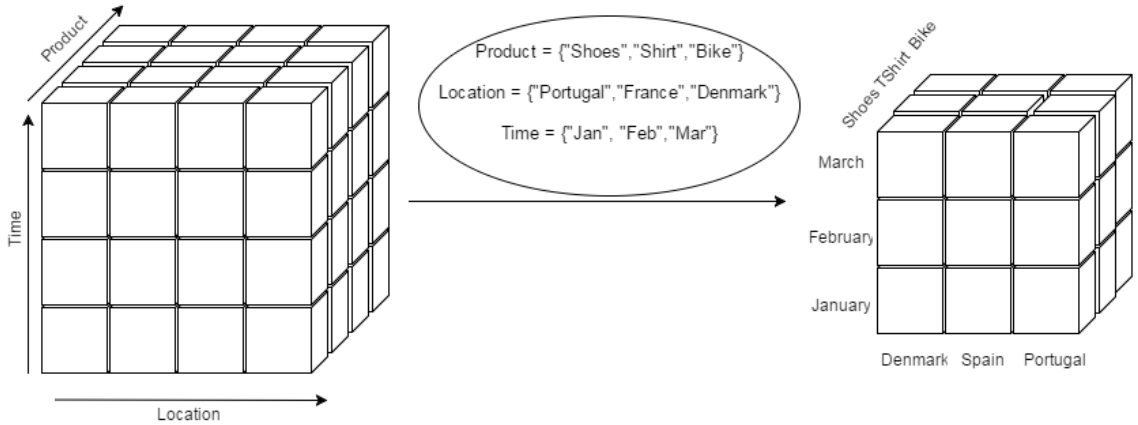


Figure 2.6: Cube dicing

2.1.5.5 Relational Multi-Dimensional Model

Relational databases managements systems (RDBMS) provides an approach to implement the conceptual model, basing on the derivation of relational models like star-schema. These models are comprehended by fact and dimension tables that captures the logical properties and provides the basis for storing and querying the multi-dimensional data.

Fact-Table As depicted on 2.7, the fact-table is the central entity in multi-dimensional schema's. It is constituted by registers that are structured, on one hand, by a set of numerical attributes, and on the other hand, by the composition of dimension foreign keys that, all together, forms the composite primary key of the fact table and identifies uniquely a business event.

The numerical attributes represents the business measures, that are complemented with the dimension keys. The latter is used to join with dimension tables, to contextualize the measures. Furthermore, the dimension keys defines the granularity of the fact table and tell us the what the scope of the measurement is [KR02].

For instance, given a range of registers of sales over the year of 2014 and 2015, analysing the data aggregated by year and month results in a fined grained data-set, when compared to analysing the data aggregated only by year.

Dimension Table Dimension tables comprehends the textual and key attributes that gives context to the facts through joining operators with the fact table. This entity can, also, embed hierarchical structures to enable the analysis of the facts over different dimensional levels.

Facts and Dimensions As illustrated on the figure 2.7 the fact-table is joined together with dimension entities, forming a star-like structure, often called a star-schema. The fact-table represents the central entity and expresses many-to-many relationships between dimension entities.

This relational modelling strategy provides the following benefits [KR02]:

1. Leverages the simplicity;
2. Introduce performance benefits;
3. Accommodates changes;
4. **Enables a complementary nature between facts and dimensions**.

In particular, a way to think about the complementary nature between facts and dimensions is to see them translated into a report [KR02]. In the the figure 2.8 resides an example, extracted from [KR02], consisted by a fact-table containing data of business related events, namely the daily sales information, and surrounded by contextual dimensions like Date, Product and Store. As the example illustrates the dimension attributes supply the report labeling, whereas the fact-tables the numeric values.

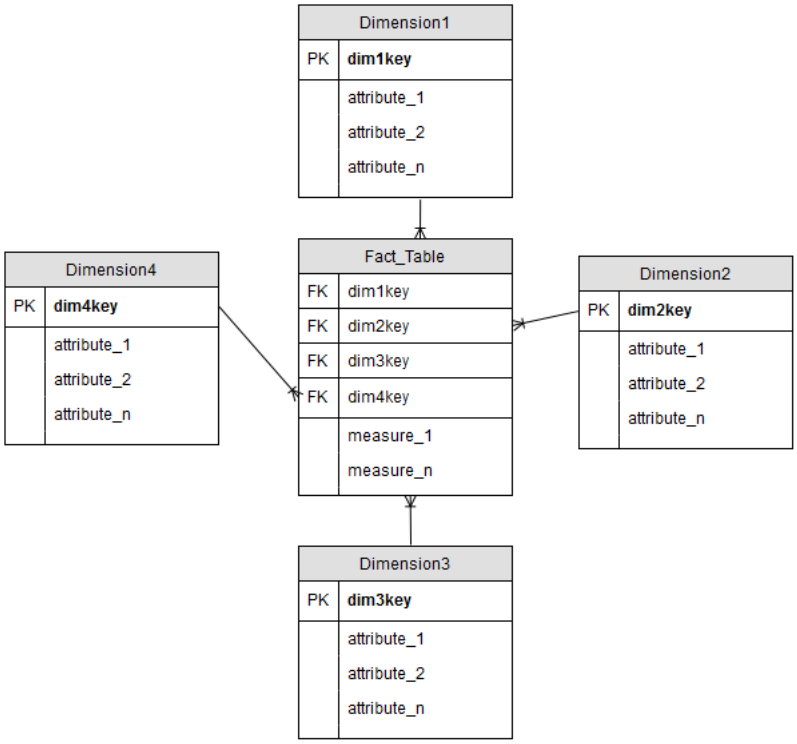


Figure 2.7: Star schema generic example

2.1.5.6 Data Access layer

The data access layer provides access to the data-warehouse and support users in their decision-making activities with the manipulation of the multi-dimensional data [Pow].

An application belonging to this layer can be a pre-defined or parameter-driven report, interactive applications like dashboards, advanced tools for visual analysis or data mining [KR02].

- **Reports:** Parametrized or pre-defined snapshots of information in a readable format. The objective of a report is to present detailed information related to a specific subject, for analysis purposes [Her]. Reports are deployed manually by IT departments, or scheduled for automatic generation and deployment;
- **Dashboards:** Dynamic information in a readable format. The objective of dashboards is to provide a comprehensive overview of contextual and measured information by presenting charts, maps, prompts for filtering, interactive features to navigate information and key performance indicators for goal tracking [Her];
- **Analytical Widgets:** A widget is a miniature application view that can be embedded in other applications[4]. A web-widget for instance, is a mini application that can be integrated on a web page. An analytical widget, is, therefore, a small and embeddable application for analytical purposes.

[4]http://developer.android.com/design/patterns/widgets.html

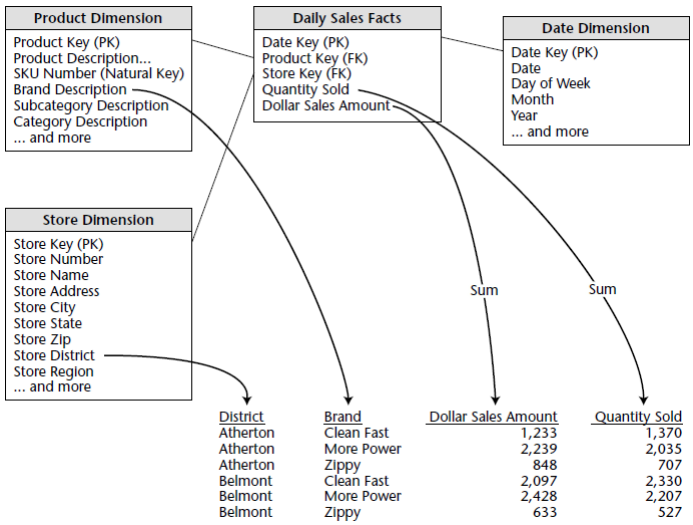
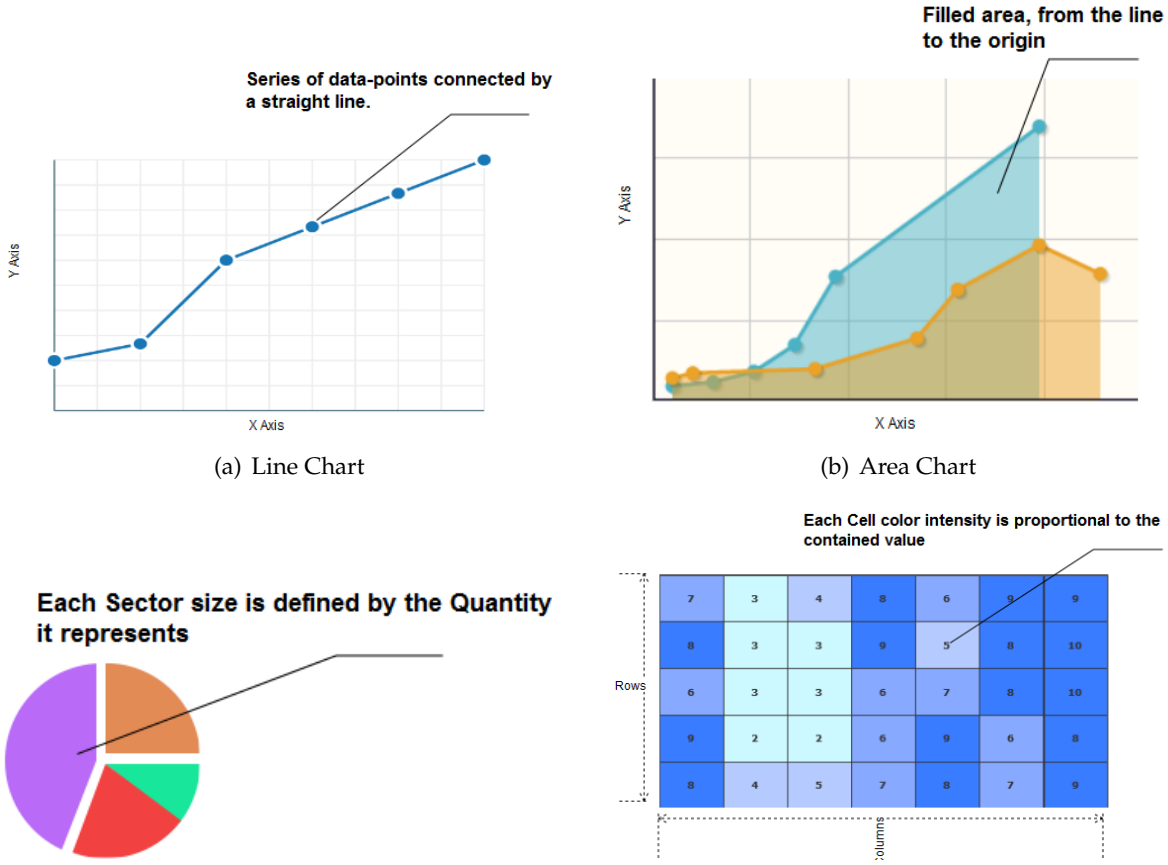


Figure 2.8: Dragging and dropping dimensional attributes and facts into a report. Image and example taken from [KR02].

Data visualization Visualization provides graphical representation of data, algorithms, processes. In particular, the visual representation of data provides an effective way to present raw data, leveraging the user visual sense. The common components used for data presentation are charts, table and maps.

Some examples of graphical data representations are:

- **Line Chart:** As illustrated on 2.9(a), it is represented by a series of data-points connected with a straight line, this type of charts are most often used to visualize data that changes over time;
- **Area Chart:**As illustrated on 2.9(b), it has the same concepts of the line chart, only it fills the area between the line and the origin Y value;
- **Bar Chart:** A bar chart presents grouped data with rectangular bars with lengths proportional to the values that they represent;
- **Pie Chart:** As illustrated on figure 2.9(c), it is a circular chart divided into sectors which is proportional to the quantity it represents;
- **Heat Map:** As illustrated on 2.9(d), it is a two-dimensional representation of data where the individual values contained in the cells of the matrix are represented as colors. The intensity of the color changes, dynamically, based on the range of the values;
- **Pivot Table:** A pivot table is an interface which enables the data analysis by turning a data-set into a summary table. Among many other operations, a pivot table can give the totals, count and average the data.



(a) Line Chart

(b) Area Chart

(c) Pie Chart

(d) Heat Map

Figure 2.9: Examples of graphical data representations

Key performance indicators A metric is a quantifiable attribute that can be measured to monitor business aspects, [Kau10] defines a metric as a "a quantitative measurement of statistics describing events or trends (...)". These attributes are typically located in the fact table.

Key Performance Indicators (KPI) is based on metrics and the strategic objectives defined by the organization. For [Kau10], the word "objective" is "critical to something being called a KPI". Thus, it is oriented towards a specific goal, to help business users to track performance. Hence, a KPI should be:

- **Readable:** Results should emphasize the meaning with versatile graphical visualizations [FHK09];
- **Enriched with context:** Aligned with readability, "The indicator should be enriched with relevant context information", in order to leverage the interpretation [FHK09];
- **Comparable:** Should be possible to analyze the interrelationships between different KPIs.

Mobile Business Intelligence The term Mobile BI is not recent. For some years that ubiquitous applications are used alongside with BI. However the devices were more limited than nowadays, reporting and analysis tasks were, therefore, more constrained. With the increase of mobile device features, this topic gained prominence [AH10].

With reference to the features and capabilities that mobile devices offers nowadays and the concepts of Business Intelligence, the definition of Mobile BI aligns with: "The capability that enables the mobile workforce to gain business insights through information analysis using applications optimized for mobile devices."[VS13]

Mobile BI applications resides on the Data access tools layer. Typical usage models of this applications on BI are [AH10]:

- **Alerts**: Alerting of events;
- **Push reporting**: On demand or scheduled report deployment;
- **Pull reporting**: The users specify the type of data they want for analysis.

2.1.5.7 The users

The user community that interacts with corporate data is vast, from technical users to dashboard end-users. Many classifications are possible to depict, driven by dimensions like role, specialization area, cognitive capabilities, skills, data access and visualization needs to support their tasks.

However the categories can be generalized, covering the specific characteristics of each user. For instance, Kurt Schegel from Gartner, a marketing research company for information technologies, provided on [Sch] a study where it classified two generic types of users: Information consumers and producers.

Information consumers is a category that comprehends the users who gather information through operational and analytical applications, namely dashboards and reports, to support their activities [IW11]. Most of the time, producing content for reporting and analysis is not a priority because this users do not have the time, experience or the need to produce and analyse information for decision making [FHK09]. An example of consumer could be an executive who spends a good portion o his time away from his desk and wants to continuously keeping track about the performance indicators from his company with pre-defined dashboards or reports[Eck11].

Information producers, on the other hand, goes beyond consumer needs. They create, maintain and analyze data to acquire deep knowledge about a certain domain and publish content to consumers. They basically represents the power users of BI applications, that is, someone whose computer skills are better than the average. This kind of users goes from collaborators capable of designing reports based on pre-defined data-sets, to users with technical and scientific knowledge to build complex analysis.

Users and the business questions Regardless the kind of user, one important aspect is the format of questions that users, in general, may want to formulate.

For instance, questions that involve standard aggregation functions to provide data summaries: "Which products, were most frequently sold in 2015? Provide a top 10", or "The average profit by product in the year 2015". In other scenarios the user needs to analyze results that are not directly addressed by the measures, these type of questions would require the definition of custom fields, for example, arithmetical operators between one or several measurable attributes, logical functions like comparison between one or several measurable attributes, statistical functions and custom scripts. The latter is useful, for example, to monitor a KPI with the indicator from the current versus the homologous year and to check if it is below a given threshold by mean of alerts. Users can also formulate questions that would require easy data navigation. For example, drilling the information, by zooming in and out at different levels of dimension hierarchies, for instance : "Products sold by year, month or day". Slicing the information by means of filtering, "How much was the profit, on day 24/12/2015?".

On other scenarios, it is possible to encounter questions that, despite of involving summaries over the data, requires specific systems to deal with specific requirements. For example, a continuous data-flow processing mechanism to provide responses in real-time:"Which products, were most frequently sold in the last 10 minutes, with a refresh interval of 10 sec? Provide a top 10". Other questions requires advanced algorithms, for instance forecasting, in order to answer questions like: " Given the today profit, how will it be for tomorrow?". Another example, would require geo-spatial features, for example: "In which areas the product is sold more often?"

2.1.5.8 Concepts Model

In the figure A.1 is depicted a model describing, graphically, the concepts of the domain and some features that BI data access applications provides. For sake of space, the figure is located on the appendix A. The foreseen solution will be aligned with some of those concepts to express the necessary configurations to map data summary questions with the (semi) automatic generation of mobile analytical components.

2.2 Technical Background

2.2.1 Mobile Development Methodologies

The constant evolution and increasing variability of mobile technologies provides to the end-users a vast spectrum of choices that better fits their needs and preferences. On the other hand, in mobile developers point-of-view, the variability of devices poses challenges, for instance, the requirement of the application being able to execute reliably, independently of the underlying operating system, hardware, screen dimensions and resolution.

The current methodologies and frameworks, on mobile development, offers to developers different capabilities. As illustrated on figure 2.10, they usually differ on the trade-off between platform independence and maximizing the user experience by accessing the features that are exposed through the device Application Programming Interface (API). By doing that, some approaches sacrifices important features like the capability to execute regardless of the underlying platform or the powerful features that one specific API provides.

The state of art on mobile development, currently comprises the following methodologies: Native Development; Cross-Platform Development; Web Development; Hybrid Development

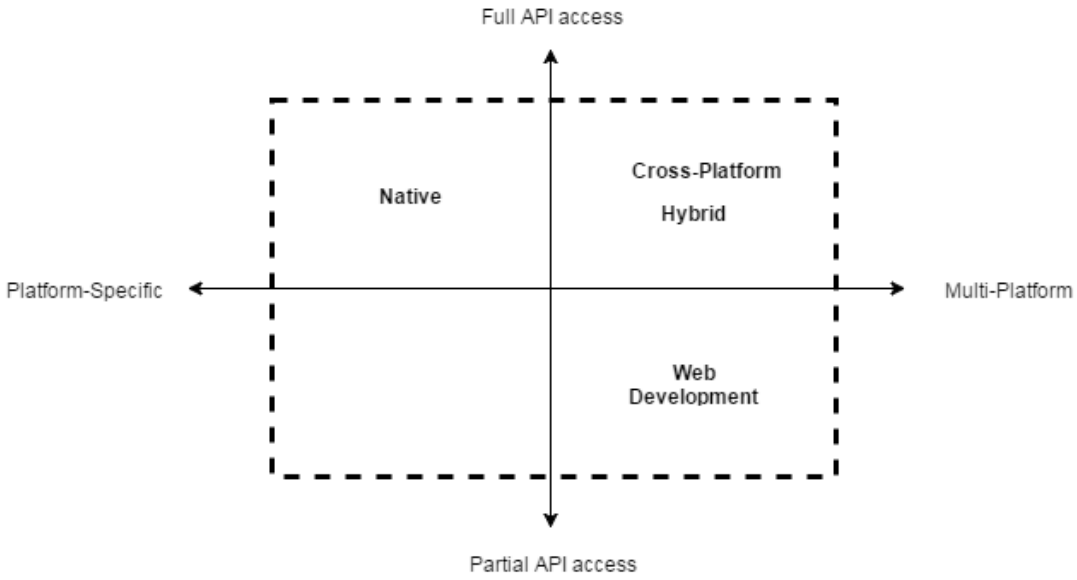


Figure 2.10: Mobile development approaches [OE15]

2.2.1.1 Native Development

This method of development is oriented to a specific platform. Based on integrated development environments (IDE), the development is driven by the supported language of the device operating system, for instance Java for Android platforms, C Sharp for Windows, Swift for IOS. The construction of a multi-platform application is, therefore, more expensive as it requires to organizations recruit various specialized developers with a broader technical knowledge related with the different platforms.

2.2.1.2 Web Development

It is essentially a web application, based on the web development standard technologies. It runs on mobile device browsers, as such it is granted to be platform independent. However, this type of applications have significant limitations regarding that it has limited access to device platform API.

2.2.1.3 Cross-Platform Development

This type of development solves the multi-platform challenge by disposing frameworks offering a software development language that is independent of the mobile platform. The independent code is then compiled to generate the platform-specific archives for installation on devices, offering similar native capabilities. This way, the developer is abstracted from learning each programming language. However, the cost of developing an application, with this approach, can be expensive if it requires a lot of customization, because each platform has it unique style and flexibility. Therefore, it is still required some knowledge of each platform API in order to leverage the user experience.

2.2.1.4 Hybrid Development

Hybrid approach takes the best from web and native-development. It solves the multi-platform challenge by applying the notion of wrapping a web application in a shell that bridges the API of the device to the browser. Thus, despite of being a web application, it offers full access to the API, offering the same features as a native one. However, because it still is a web application, it fails at delivering the same performance and pleasantness as truly native applications.

2.2.2 The Target Platform

The target platform is a hybrid mobile application for Mobile Business Intelligence. It is based on the Ionic front-end framework, Angular JS as the client-code, Highcharts as the chart library, SQLite as the local data persistence and Cordova as the native bridge.

2.2.2.1 Related frameworks and technologies

Angular JS (Version one) [5] Is a Javascript framework for dynamic web apps. Essentially, based on its core features, it extends HTML syntax to makes the web-page more responsive to user interaction. It is based on separating the presentation layer with the business logic, following a kind of Model View Controller (MVC) or Model View View Model (MVVM). The author on [Min12], discusses that, the Angular JS framework does not implement this patterns in the traditional sense, rather the developers of this framework document the pattern as a Model-View-Whatever (MVW), in the sense that the developer can choose which Model-View pattern to use, based on their preferences or requirements.

Example of documented core features of Angular JS are [6]:

- **Controller:** Javascript functions that are bound to specific scope objects;
- **Scope:** Objects that represents the model. They are the glue between the controller and the view;

[5] https://angularjs.org/
[6] https://docs.angularjs.org/guide

- **Data Binding:** Is the automatic data-synchronization between view and model. This way, the view is the projection of the model, at all times. The Angular JS framework provides a strategy for synchronization where any changes on the view are reflected on the model and any changes on the model are propagated on the view, this approach is documented as Two-Way Data Binding;

- **Services:** Singleton-objects that are bounded and instantiated in the application. For example the $http [7] Service to communicate with remote HTTP servers;

- **Directives:** Directives are HTML markers, like elements, attributes. It can be used to create new custom page components. Angular JS comes with a set of built-in directives, like the ngIf[8] directive, that recreates or removes a portion of a HTML element, depending on the result of evaluating a conditional expression.

Apache Cordova [9] It is a fork from the Phonegap project. It is an open-source mobile development framework, that allows the development of web applications using standard technologies (HTML5, CSS3, Javascript) and the deployment on mobile devices, regardless of the platform. Essentially, it acts as a wrapper that bridges the web applications with the API of mobile devices.

SQLite [10] It is library that implements a lightweight SQL database engine. It can be easily integrated in native wrappers like Cordova with the support of plug-ins and accessed with Javascript APIs.

Ionic [11] It is a framework aimed to hybrid mobile applications user interface development, with HTML5. The goal is to provide to users a look-and-feel of native mobile applications, when actually is web-based. It relies on Angular JS for core functionality and on native wrapper technologies like Phonegap and Cordova.

Highcharts [12] It is a library of charts written in pure JavaScript, offering an easy way of adding interactive charts to web applications. It has a wide variety of charts configurable and customizable with Javascript Object Notation.

PivotTable.js [13] It is a Javascript Pivot Table library with drag'n'drop functionality. It is used to display the information in a tabular format, besides, providing interactive features.

[7] https://docs.angularjs.org/api/ng/service/$http
[8] https://docs.angularjs.org/api/ng/directive/ngIf
[9] https://cordova.apache.org/
[10] https://www.sqlite.org/
[11] http://ionicframework.com/
[12] http://www.highcharts.com/
[13] http://nicolas.kruchten.com/pivottable/examples/

2.2.2.2 Target Platform: The first prototype

The Mobile BI prototype implemented by the company is a mobile application for data monitoring. The motivation to develop such application was aligned with the need of providing a customizable and extensible mobile solution where they have full control regarding which analytical components are delivered to the clients.

For this purpose, the company established the following goals:

- Capacity of providing the visual information in a quick and efficient manner;
- Synchronize the corporate information with a remote server;
- Be capable of execute, regardless of the platform.

To realize these goals, the prototype was implemented as a hybrid mobile application and its technological stack is formed by the web-development technologies like HTML5, CSS3, Javascript, front-end frameworks like Ionic and AngularJS (Version one), Highcharts library for data-visualization, backstage core components like the Apache Cordova wrapper and SQLite Database Management System for local data persistence and management. The architecture is illustrated on figure 2.11.

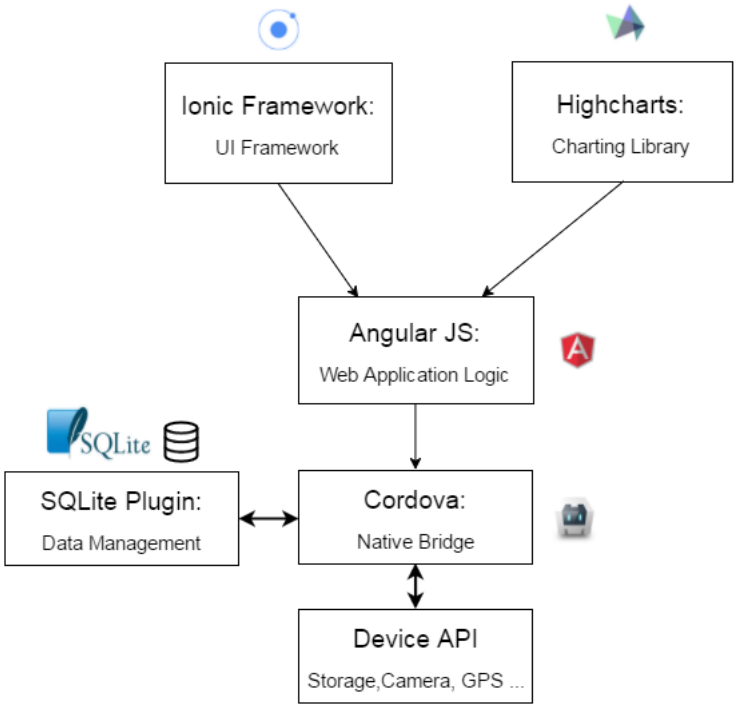


Figure 2.11: Mobile BI Architecture

In addition, outside of the core scope of the Mobile BI project, it was implemented a server to synchronize the data with the target-platform.

The conceived Mobile BI application presents to the end-user a set of navigational and interactive views composed by pre-defined analytical components, as represented

on figure 2.12. The user interacts with the dashboard by choosing which data to present, based on drop-down menu filters.

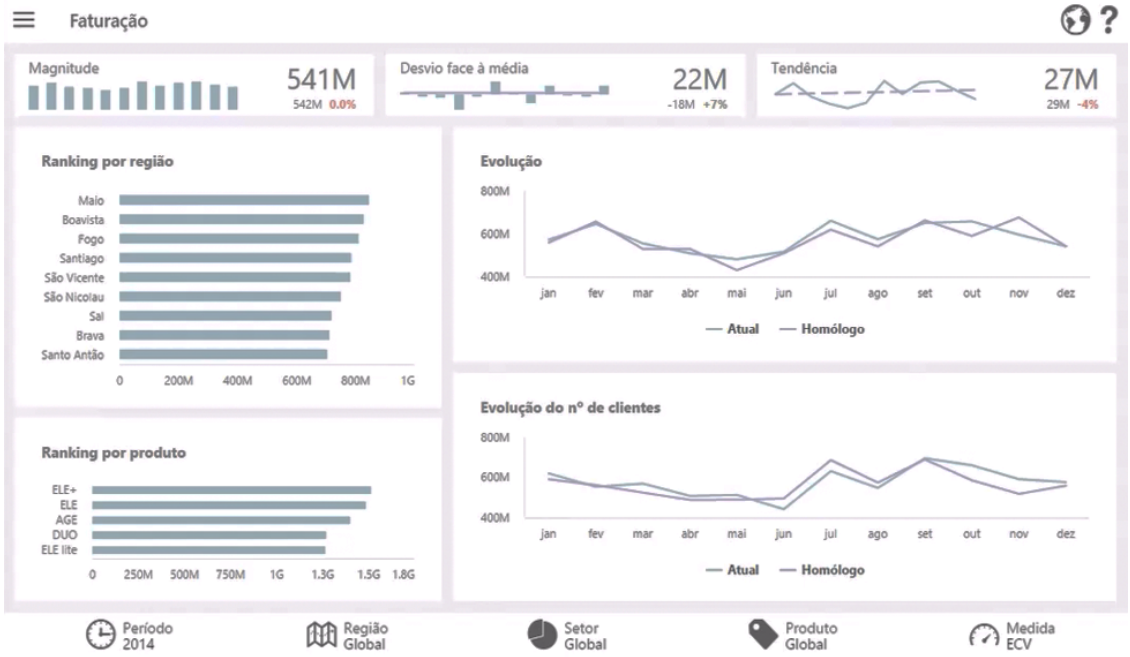


Figure 2.12: Mobile BI Dashboard example. The end user does not have the features to generate new widgets nor to configure the dashboard presentation

2.2.3 Modelling tools

Modelling workbenches provide a simplified way to develop DSLs. There is a wide spectrum of choices, among which: GME [14]; MetaEdit+ [15]; AtomPM [16]; EMF [17].

2.2.3.1 EMF

EMF, for instance, stands for Eclipse Modelling Framework and is a modelling framework and code generation facility to build tools and other applications based on models. The Ecore is the core meta-modelling notation and it is used to describe EMF models.

With the support of Graphical Modelling Framework (GMF) [18], is provided a generative component based on model-driven approach and a run-time infrastructure to build graphical editors based on EMF.

[14] http://www.isis.vanderbilt.edu/projects/gme/
[15] http://www.metacase.com/products.html
[16] http://www-ens.iro.umontreal.ca/ syriani/atompm/atompm.htm
[17] https://eclipse.org/modeling/emf/
[18] http://www.eclipse.org/gmf-tooling/

2.2.3.2 Epsilon

Epsilon [19] is a family of task-specific languages to manage models (Model validation, code-generation, model-to-model transformations).

Example of task-specific languages are:

- **Epsilon Object Language (EOL):** The common language of Epsilon, used to manage models. It provides common features of javascript and OCL to create, query and modify EMF models;
- **Epsilon Validation Language (EVL):** It is a model validation language built on top of EOL;
- **Epsilon Generation Language (EGL):** It is a template-based language, built on top of EOL, for code generation and other textual artifacts, from model specifications.

Despite being extensively used with EMF, it is not bounded to this framework. In particular, Epsilon can be extend with the support of additional technologies, for example XML or JSON. The Epsilon Model Connectivity was conceived for this effect. The Epsilon developers defines EMC as an open model connectivity framework which developers can extend with support for additional types of models/modeling technologies by providing respective drivers.

2.3 Summary and discussion

In this chapter we described the fundamental concepts which are the support for the solution development. Besides that, such concepts are crucial to read the document and to understand the solution.

We started by describing some theoretical concepts such as MDD, DSLs and SPL. Then, we performed a domain analysis encompassing the concepts of BI, which ended with a conceptual model - A.1 - describing those concepts, graphically. Lastly, some technical aspects were presented including the detailed presentation of the first prototype, introduced in the first chapter - 1.1.

Technically, the solution should reside in the extension of the first prototype in order to address the user empowerment issue. The MDD, DSLs and SPL topics, together, aligns with and tackles this issue by providing the concept of domain-specific abstractions and automation to shield the end users from aspects related with GPLs, thus leveraging their productivity. Therefore, these theoretical concepts can be and will be used on this matter.

Nonetheless, before we start to delve into the theoretical and technical details of the foreseen solution, a study comprising the state of the art on the user empowerment issue in BI must be conducted. The results of such study will inspire us to identify what kind of solution aligns better with the context of the problem. The next chapter will cover this point.

[19] http://www.eclipse.org/epsilon/

3

The User Empowerment on Business Intelligence

In this chapter is described the state of the art related with the user empowerment issue on Business Intelligence. Such related approaches on this topic will clarify the kind of solution which is better aligned with the context of the problem.

As seen previously in the BI domain analysis - 2.1.5 - reports and dashboards are some of the applications that provides access to data. However, constructing them, traditionally requires some level of technical expertise that some users do not have. As aforementioned, this affects both types of users: the technical and the non-technical.

Mobile BI can be considered as an approach to solve the user empowerment issue. In the sense that, end users are not chained to their working place to monitor their corporate information. However, this capability, by itself, does not solve the problem regarding the need of constructing new analytical components, on demand.

This is were the Self Service BI acts. It is a set of facilities that follows a "Do It Your Self" paradigm. Empowering business users to answer their information needs, on-demand, becoming more "self-reliant and less IT dependent" [IW11].

3.1 Categorization

The technological state of the art related this topic is immense. A categorization is depicted on figure 3.1. It explains the category of tools oriented to self-service reporting and data-analysis.

In 3.1.1, 3.1.2 and 3.1.3, explains the categories of tools that provides the self-service

capabilities. Furthermore, in 3.2, is given technological examples regarding those categories.

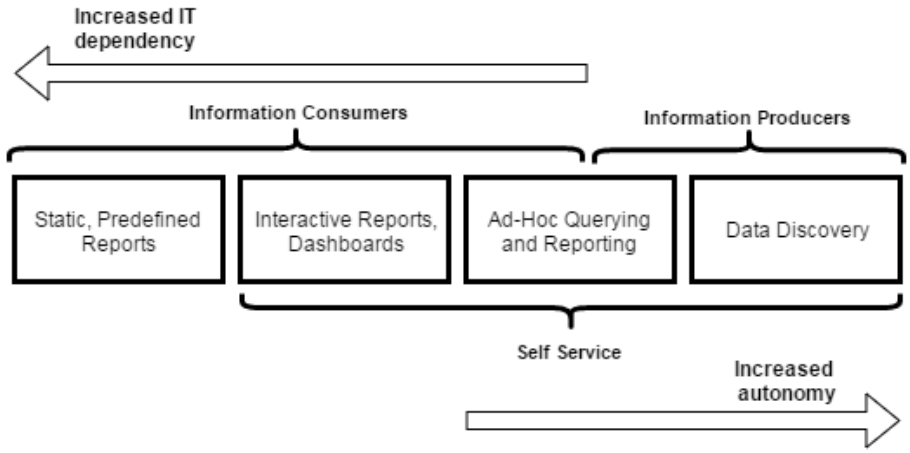


Figure 3.1: Gartner on self-service tools and type of users. Adapted from [Sch]

3.1.1 Interactive Reports and Dashboards

Interactive reports and dashboards delivers some empowerment level, as users can manipulate the data that is fed to visual components by means of filtering and navigation features. Nonetheless, sometimes, there is a need of deeper analysis to answer questions not initially addressed by reports or dashboards. This, brings the need for facilities which facilitates the construction of new analytical components.

3.1.2 Ad-Hoc querying

Ad-hoc querying is a Business Intelligence model that solves the aforementioned problem. It empowers the domain users by enabling them to answer specific questions, on-the-fly, and in a self-dependent manner. Example of facilities that fall on this category are the Ad-Hoc reporting tools, used for instance, to create a short report to analyse the reported information, or to complement an IT pre-defined report, in order to gain a deep understanding of a specific evidence.

However, this model is usually limited to the BI architecture and standards instantiated by technical people. This way, the empowerment is not sufficient if the user wants full control over the data, from its source to visualization, for example to easily blend information from multiple sources and present it in a single dashboard.

3.1.3 Data Discovery

Data discovery is a set of analytical software that provides easy access and visualization of data, by extending the ad-hoc model with the ability of easily access multiple sources of data and provide an interface with features that enables the end-user to construct simple to complex visualizations.

The main advantages of this set of tools resides in the possibility of mashing up different sources of data and extract knowledge for ad-hoc analysis, without the full support of complex processes from IT departments to extract transform and load the information on presentation areas.

In theory, this family of tools are the ones which, typically, provides more freedom. However, in practice, it over empowers the business users, resulting in tools which are too difficult to use correctly by some users, or which does not complies with governance rules of the organization.

3.2 Example of Tools

The state of the art in Self-Service and Mobile BI is vast. Most of the existing solutions unifies this two worlds by providing back-office platforms for construction and deployment of the analytical components.

3.2.1 SAP Web Intelligence

SAP Web Intelligence (WEBI)[1] is a self-service tool for ad-hoc reporting. It belongs to Business Objects (BO) [2], a former software company, that now belongs to SAP group.

It is web-based and without any programming knowledge, users can construct and export reports with an interface optimized for usage in desktop computers. It provides two main modules, known as Query Panel and Report Editor, respectively depicted on figure 3.2. The Query Panel goal aligns with concept of the complementary nature between facts and dimensions studied on the Domain Analysis (section 2.1.5.5). The goal is to pre-model a data-set with drag-and-drop actions over blocks that represents the attributes from the data schema, into a drop-down area which receives the dropped blocks. This interaction generates a data-set for further exploration in the Report Editor. The latter enables the users to create fast reports based on "for this purpose" (Ad-Hoc) queries over the pre-defined data-set and conforming to the business terms defined in the semantic layer, denominated as Universe.

The semantic layer is a configuration module that manages information about the connection with the data-sources, the relying data-models and its notations, namely the names of each entity attributes. These notations can be translated onto simplified and common terms, related with the business area. This brings advantages because the business user is most likely to be familiar with those terms. Nonetheless, due to the technical concepts and knowledge that this task requires, the configuration of the semantic layer is assigned to the IT staff, and depending on the size and complexity of the requirements, it can be a slow task.

[1] http://go.sap.com/product/analytics/bi-platform/web-intelligence.html
[2] https://pt.wikipedia.org/wiki/Business_Objects

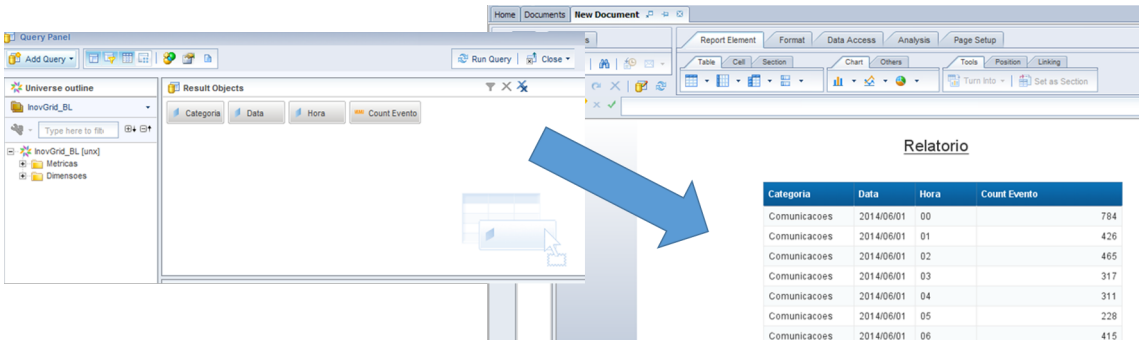


Figure 3.2: SAP WEBI. First the end user interacts with Query Panel to model the data-set. Then the data is available to explore in the Report Editor

3.2.2 Tableau

Tableau Desktop[3] is a self-service data-discovery tool. It was born in an academic environment and was evolved from a system called Polaris [HM07]. The goal of the Polaris system was to provide an easy exploratory visualization interface to let users navigate data in multi-dimensional relational databases. It is now commercialized by Tableau and its features are now extend by Tableau Desktop, one of the reference tools for self-service BI. Its powerful features of data manipulation and visualization provides to users a platform suitable for basic to complex visual analysis, not just a system to create high quality visual components [HM07]. The main module of Tableau Desktop is the visual query language, denominated as VizQL. It is the run-time module and conforms with an abstraction layer that captures the meta-data required to access the data-sources and to describe the data model, classifying its attributes, based on heuristics, as measures or dimensions. As represented on figure 3.3, the interaction with the visual query language is made with drag-and-drop actions over the blocks and into shelves that receives the dropped attributes, from which is generated the data-visualization.

3.2.2.1 Tableau Mobile

Tableau Mobile[4] enables users visualize and interact with content, on the go. It provides access to the data-visualizations deployed on the server.

3.2.3 SAP Lumira

SAP Lumira[5] follows the same paradigm of Tableau. It is a desktop-based, data discovery tool. Based on a explicit sequence of steps, it enables users to acquire, manipulate, visualize data through charts or maps and compose dashboards that can be shared with other users.

In order to enhance the user-friendliness, the work flow is guided by four steps:

[3]http://www.tableau.com/products/desktop
[4]http://www.tableau.com/products/mobile
[5]http://go.sap.com/product/analytics/lumira.html

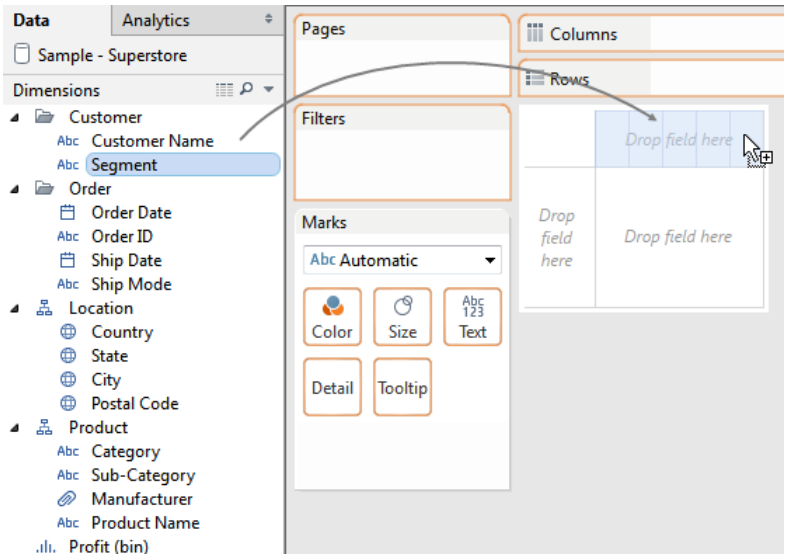


Figure 3.3: Tableau Desktop. Dragging of fields to property shelves. This interaction generates a visual component

- The first one is data acquiring. This tool provides a set of connectors that enables users to gather data from multiple data-sources;
- Data prepare is another step of the flow. This phase enables users to enrich the acquired data-set before building the visualizations;
- Then, users can easily create visualizations based on a set of pre-defined analytical components like charts, tables or maps. To populate this components, Lumira also provides a visual query language, based on drag-and-drop actions over the data-schema attributes;
- The final step is sharing the information. The user can compose dashboards and reports to share with the organization.

3.2.3.1 SAP BO Mobile

SAP BO Mobile allows the visualization, on mobile devices, of content built on back-office and deployed on server.

3.2.4 Datazen

Datazen[6] is a Mobile Business Intelligence solution that was built to enable a rapid publishing of BI content such as, KPIs and dashboards, at the same time as granting a good user experience, independently of screen dimensions. Despite of not being, officially classified as a Self-Service BI solution, it offers some degree of freedom to users by integrating an administration panel for KPI construction and a publisher platform for dashboard designing and deployment, which is accessible on back-office context, on desktop devices.

[6] http://www.datazen.com/

3.2.4.1 Construction of Key Performance Indicators

The construction of KPIs is a process which is performed in the control panel of the core service. It requires administration permissions and some technical know-how.

Datazen defines a KPI as an entity that contains the following properties:

- **Current value:** This is the main attribute, as it contains the value that describes the performance. The value has an associated type, for instance Currency;
- **Goal:** This value is used for comparison with the current value, it enables to end-users to visualize how far they are from a defined goal;
- **Status:** Defines a dynamic style event alerting when the indicator reaches some threshold, for instance to notify if the current value has exceeded a specific goal threshold.

This variables can be instantiated manually with static values or generated from textual query specification, for example in Structured Query Language (SQL) syntax. Each KPI can be associated with one or more dashboards, enabling end users to visualize the indicator.

3.2.4.2 Construction of Data-Views

Based on the specification of textual queries at back-office, is possible to define an intermediate layer of data-views. This views can be parametrized when designing the dashboard, shaping the scope of information to be presented by the widgets.

3.2.4.3 Datazen Designer

Datazen Designer is a back-office platform for dashboard construction, on desktop devices. The main features provided by this component are:

- **Layout Designing**: As represented on figure 3.4, based on drag-and-drop actions and a grid layout, users can place widgets on grid cells and adjust its size dimensions. Each draggable widget has a specific role on the dashboard. For example, the Navigator is used to store the parameter values to filter the presented information. The Gauge is used to show indicators, Chart for data visualization and Map to present and to enable interaction with spatial data.

  Each widget has its visual properties, like title, sub-title, type of numeric values, series visualization (bar,line,area). These are defined with the selection of options in menus.

- **Data Connection**: In this section, the user specifies what data is presented in each widget, by importing the meta and the actual data from an excel file or from a connection with a database. Optionally, parametrized data-views can be used.

Figure 3.4: Dashboard construction with drag-and-drop actions over pre-defined widgets and into the dashboard panel

3.2.4.4 Datazen Viewer

The viewer platform connects to Datazen server and provides to users, read access to published dashboard and KPIs, in desktop or in mobile devices.

3.2.5 Roambi

Roambi[7] is another Business Intelligence solution designed for mobile devices, it offers features that leverages the user interactivity. It also offers some level of freedom to users with a back-office content publishing web-platform, on which is incorporated a user interface optimized for running on desktop or mobile web-browsers, on which the users can design and publish dashboards for usage in mobile devices. For this purpose, it disposes a set of configuration dashboard templates. These templates are configured on the publisher and then deployed and accessible on mobile devices.

Examples of dashboard templates are:

- **Pulse Template:** The concept is to present a master-detail panorama. This idea is realized based on the presentation of interrelated KPIs. In addition, each KPI is related to a detailed dashboard that presents the indicator evolution over time. The configuration of a pulse dashboard follows a style of a fill-in-the-blanks configuration, based on the meta-data attributes, extracted from the data-source;

- **Card-View Template:** Editor with a grid interface, enables users define and place

[7] https://roambi.com/

charts, tables in the grid locations. For each component, the user specifies the visual properties, like series, category dimensions by selecting the attributes from the data-set.

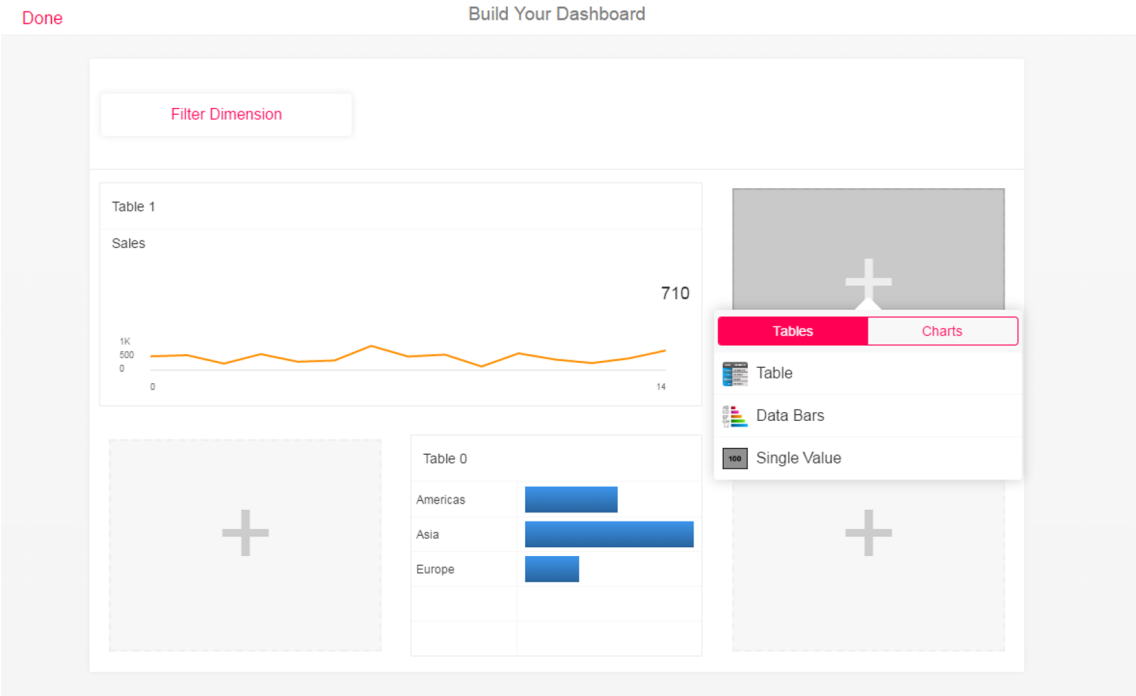


Figure 3.5: Card-View template configuration

3.3 Summary and discussion

In this chapter we presented examples of technologies towards the user empowerment issue on the Business Intelligence domain. Examples were provided based on the categorization depicted initially, wherein the difference between each tool resides on the balance between IT dependency and the user autonomy. As common characteristics, all solutions recognizes the potentiality of mobile BI applications, providing content publishing for mobile devices.

Given the tools described, the ones that provides more autonomy from the IT staff-Tableau Desktop and SAP Lumira - typically have the higher expressiveness in terms of data manipulation, being able to cover a vast spectrum of user visualization needs. The majority, relies on the usage of visual query languages, in order to leverage the visual sense of the user, and on data-blending features, to analyse data from different sources of data, with the minimum IT involvement as possible.

On the other hand, tools like Datazen and Roambi, despite of not providing the expressiveness of Tableau, SAP Lumira or SAP WEBI, they focus on providing a simple solution which enables the user on quick analysis on mobility contexts. As seen, these two tools provides back-office platforms to configure and design the analytical dashboards

for mobile deployment.

It is important to note that, despite the differences between the technologies analysed, they are not mutually exclusive, but instead, they usually can be used to complement each other. For instance, tools like Roambi or Datazen are better suited for business users who spends most of their time travelling and usually does not have the time nor the inclination to access detailed tools like SAP WEBI to gather simple insights. Thus, these tools can be combined, such that, technologies like SAP WEBI can be used to construct deeper and detailed analysis by BI consultants and tools like Roambi used by the business users to construct and visualize quick analysis for mobility contexts.

Hence we can conclude that, our vision is aligned with the Roambi and Datazen solutions. However, we will not focus on conceiving a final product with a closed set of libraries and features. But, instead, we will invest on a mean of cooperation between the business people and the domain experts. These both actors will, then, perform different roles towards a similar end, which is to empower the end user.

The Software Product Line can be applied to conceptually describe this idea. The product line scope is maintained by the domain experts to develop the production assets, regarding the business requirements and needs. The production assets are supplied for product development, being accessible for business people who will perform the product developer role to generate the final product, which is the analytical widget. This cooperative strategy brings advantages when it comes to situations where the client wants to be empowered, and has specific requirements which requires specific analytical components. Since the user empowerment is an important feature, the product line developer would define a production assets scope, so that, it provides assets that are aligned with the requirements, but are generic enough to be used and configured by the end user based on his preferences, with the final goal of generating his product.

This idea, also, aligns with the initial motivation of the company: To have a customizable and extensible solution where they have full control regarding which components are delivered to the clients. However, the development efforts need to be optimized, towards the conception of generic assets. The next chapter serves as introduction to explain how these aspects were addressed in order to solve the user empowerment issue.

Table 3.1: User empowerment tools comparison. Table constructed based on the tools study

Tool	*Classification*	*Data manipulation*	*Usage scenarios*	*Chart library extension*	*Mobile deployment*	*Mobile dashboard configuration*
Tableau Desktop[a]	Data discovery	Draggable fields	Advanced to simple analysis	Limited	Yes	Back office
SAP Lumira[b]	Data discovery	Draggable fields	Advanced to simple analysis	Limited	Yes	Back office
SAP BO WEBI[c]	Ad hoc Reporting	Draggable fields	Reporting	No	Yes	Back office
Datazen[d]	Mobile Dashboard design	Forms	Quick dashboard design	No	Yes	Back office
Roambi[e]	Mobile Dashboard design	Forms	Quick dashboard design	No	Yes	Back office

[a] http://www.tableau.com/products/desktop
[b] http://go.sap.com/product/analytics/lumira.html
[c] http://go.sap.com/product/analytics/bi-platform/web-intelligence.html
[d] http://www.datazen.com/
[e] https://roambi.com/

4

An *Energy & Utilties* Case Study and Solution Specification

So far, we analysed the BI domain by describing its concepts. Also, we studied the related work towards the user empowerment on BI. From here, we will dive into some generic aspects regarding the conceived prototype by explaining how the theoretical concepts were allocated and by providing an user interface specification, starting on the extended version of the Mobile BI Target Platform and ending on the Editor. Its details are then explained, starting from the modelling language design (chapter 5) and ending on the architecture implementation (chapter 6).

However, in first place, a case study is presented as running example used on this dissertation context - also used on the usability tests (chapter 7). Its description is important in order to understand the business concepts and to decide what kind of production assets are important to fulfill the business users needs.

It is important to note that, despite the production assets and the architectural components were build conforming to this specific case study, they are not limited to this specific business. Instead, they can be re-used on other contexts.

4.1 Case Study

The case study is based on the *Energy & Utilities* business area, focused on a fictitious Utility which is inspired on a real client. The products and services provided by the Utility are aligned with the supply of Electricity and Water to their customers, covered on all regions of the activity area, structured on islands, municipalities and parish.

Each customer is associated with a type of contract related to the kind of client it

represents (e.g individual or collective) and the respective periodic charge that reflects on the customer billing and the revenue of the company, in contrast with customers in debt, that affects negatively the company income. The goal is to monitor the corporate information, namely the data related to customer charging and payments, namely:

- **Billing:** Monetary value billed to the clients;
- **Debt:** Monetary value that clients owes, as a result of not paying the bills. The bigger this value is, the odds of having financial losses increases, as well;
- **Number of owing contracts:** Number of contracts that are currently owing money to the company.
- **Profit:** Related to the customer charging and payments, it is the value that was received from the clients who paid the bills. For sake of simplicity, it is the difference between the Billing and the Debt.

In this case study we also based upon the following descriptive information:

- **Periods:** Time-based dimension with monthly and annually periods;
- **Regions:** Business activity regions - In this context they are identified as islands;
- **Sectors:** Business activity sectors, namely Water and Electricity areas.
- **Product:** Product types that can be associated to a contract:
  - Electricity product;
  - Water product;
  - Dual product: The Electricity and Water services in the same contract.

The aforementioned information is projected on the data-model of the figure 4.1. Note that, giving the current architectural implementation of the Target Platform, the case study concepts and its respective data-model needed to be a subset of the real problem, so that the data-set has a coarse-grained granularity. For example, considering the data-model, it can be verified that no information is present regarding the consumers, namely, for instance, a dimensional entity called *Dim Consumers*. If we had considered to analyse each consumer of the Utility, it would result in high cost queries and storage space on the mobile devices. Therefore, the analysis are typically grouped by regions, sectors, products and periods.

About the business needs, they are strongly interested on time-base analysis, wherein they can analyse the information across different years and months. One particular analysis they are interested is to visualize the information on the current year and the previous one. Where they can see, for example, a specific event on the current year and its homologous, i.e., on the previous year.

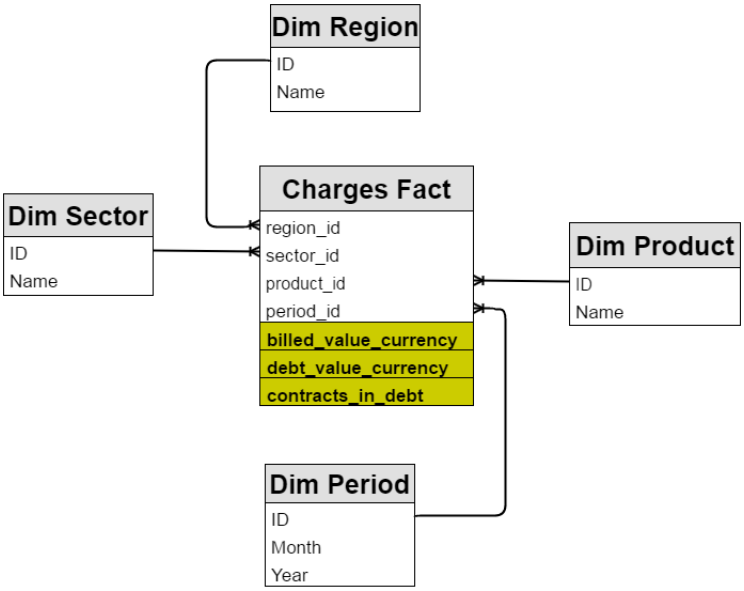


Figure 4.1: Case study star-schema

They are also interested in analysing the information across other different dimensions, namely by its activity sectors, products and the island regions. The requirements are projected on the feature diagram on the figure 4.2, wherein we can analyse the products family variability and the commonalities. As such, the widgets family required, typically need to provide analysis concerning the business information, across the provided dimensions, illustrated on different data visualization component types, e.g. Line Chart, and to enable the execution on mobile devices. To leverage some interactivity, some interactive features should be available, namely the conceptual cube operations.

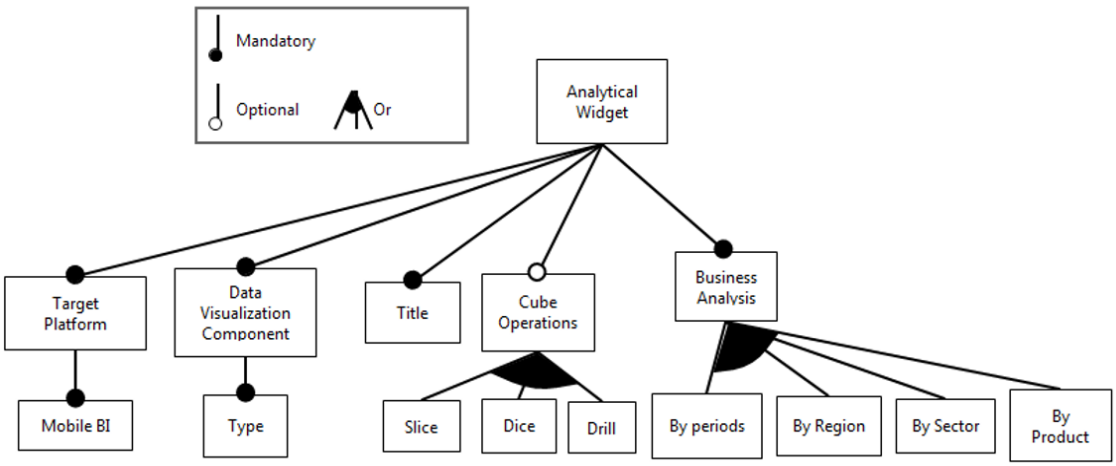


Figure 4.2: Feature Model describing the product family variability and commonalities

4.2 Solution Overview

Given the presented case study and the current version of the first prototype, a question still needs to be answered: How to empower the business users so that they can generate their own widgets accordingly with their preferences?

The solution is inspired on the SPL principle to reach a mean of cooperation between two main actors - the domain experts and the end users. Wherein, a software product line is a production capacity for a family of software products, build from a common set of production assets [Gre+04]. The domain experts will perform administrative activities by being responsible to maintain and develop the product line, to provide the production assets for end user configuration. The result of such configuration is a product - also referenced as widget - which enables the user to monitor the data.

The domain experts are the specialists of BI, understanding its theoretical and practical concepts in addition of having possible programming backgrounds. The end users group aggregates the business people, which were previously identified as the main target on the user empowerment capability. The business users group includes different levels of skill set, from people with full understanding regarding the business concepts and without any technical knowledge, to business people who despite of additionally having a technical background, still not have the expertise on the BI area.

Regardless the skill set, it is important to have an interface which rises the abstraction level, decreasing complexity and leveraging the simplicity so that the end user can configure its products/widgets.

Therefore, the expected result is the conception of a prototype, based on cooperative strategy, capable of empower the end user - as illustrated on the figure A.2 (for sake of space, the figure is located on the appendix) - wherein:

- It is conceived a back end facility which is responsible to manage the production assets supplied by the domain experts team;
- It is conceived a front-end component with a modelling language, specific to BI domain and suitable for the end users, to configure the products/widgets based on the assets deployed by the domain experts;
- It is conceived a back end core facility which, based on the MDD concepts, is responsible to read the configuration made by the end user and to generate the respective product/widget;
- It is conceived a mobile front-end component to be used by the end users (the target platform): To incorporate the generated products/widgets; To enable the dashboard configuration; To enable the end user to monitor the corporate information.

The prototype conception was based on the following set of stages - as illustrated on the figure A.3:

1. Specification regarding the user interface of the front end components;
2. Designing of a modelling language specific to the BI domain, so that, the end user can generate the widgets:
   - Modelling language abstract concepts, concrete syntax and well-formdness rules definition;
3. Architecture design and implementation:
   - Implementation of the concepts analysed on the modelling language design;
   - Implementation and technology instantiation for each architectural component, conforming to the architecture design. Definition of re-usable assets, model-to-code-generators and model validation;
4. Prototype validation.

4.3 UI Specification

4.3.1 Second Version of the Target Platform

The extent version of the Target Platform is composed by a grid dashboard where the end user selects the generated widgets. In the figure 4.3, is illustrated an example where the dashboard is empty. To add new widgets, the end user navigates to the Widget Repository by opening the menu on the top right corner.

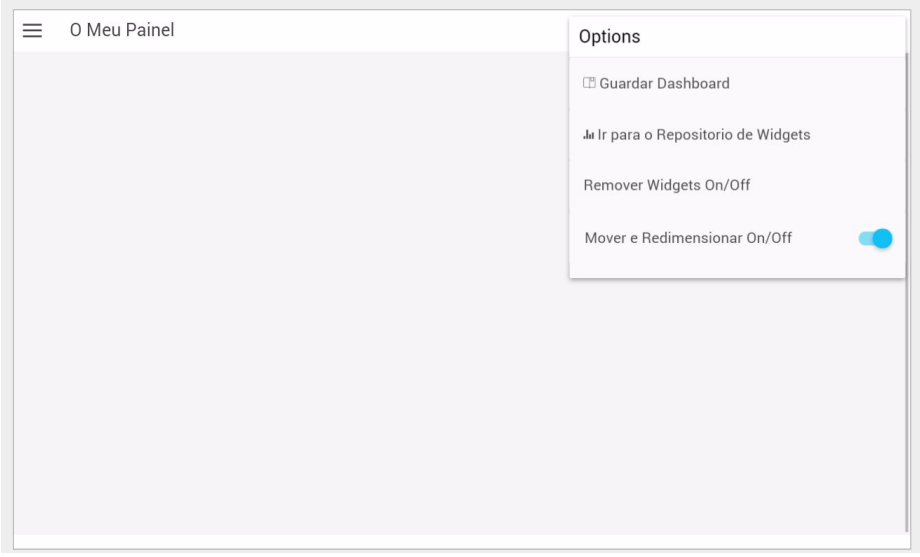


Figure 4.3: Target Platform: Dashboard for end user configuration. On the top right corner an options menu is open

On the Widget Repository view, illustrated on the figure 4.4, the list presents the widgets that are installed on the device. A widget can be selected and added to the dashboard. If the end user wants to create a new widget, he/she is required to tap the plus

icon on the top right corner in order to reach the Editor view.

Figure 4.4: Mobile Target Platform: Widget Repository, the user selects which widget he wants to visualize on the dashboard

Let us suppose the end user adds a widget on the dashboard. As illustrated on the figure 4.5, the dashboard grid system enables him/her to customize the presentation of the dashboard by dragging, resizing and removing actions over the widgets.

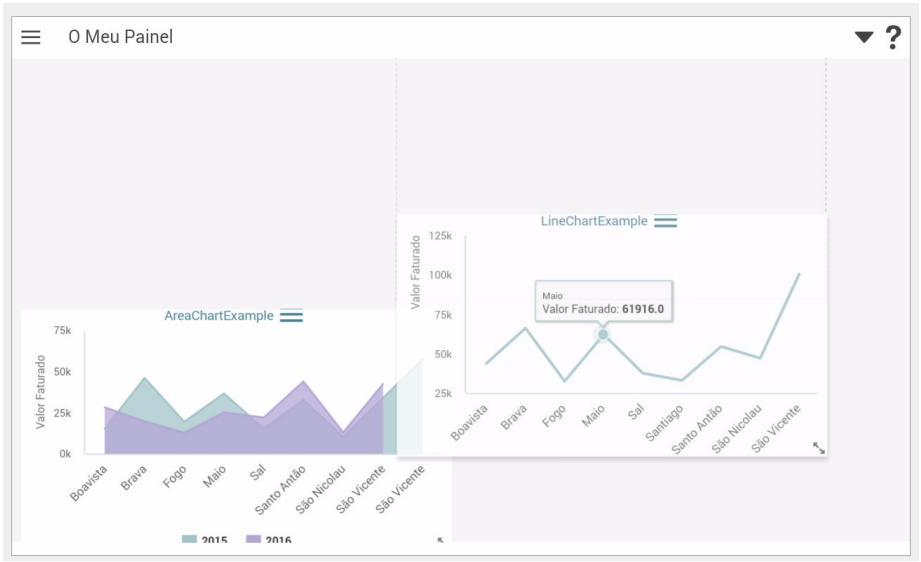


Figure 4.5: Mobile Target Platform: Dragging the widgets on the dashboard

4.3.2 Editor UI specification

Supposing the end user wants to create a new widget, the Editor view presents a set of properties related with the selected widget type. As illustrated on the figure 4.6, each property is a container, in which, can be added attributes from the meta-data.

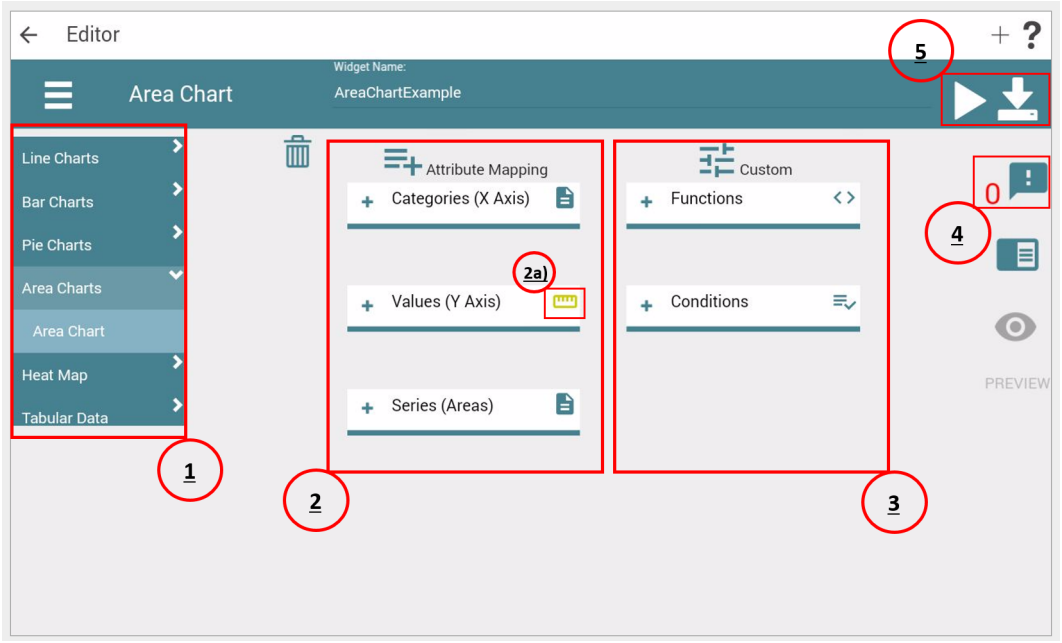


Figure 4.6: Editor View, while on mobility context

Based on the markings of the figure 4.6, the following list describes each element of the interface:

- **Marking 1**: List of widget types. Tapping on one specific widget type triggers an event, from which the Editor updates its interface to present the widget specific properties;
- **Marking 2**: Widget specific properties. On this containers the attributes from the meta-data are mapped. The attributes mapped on each container are represented as blocks - figure 4.8. To add attributes from the meta-data, the end user must tap the plus icon on the respective container. This action opens a panel as represented on the figure 4.7, in which, are listed the attributes;
  - **Marking 2a)**: On each container, an icon shows the kind of blocks that it accepts;
- **Marking 3**: Common properties. On this containers the goal is to extend the widget by customize or filter the information to be presented;
- **Marking 4**: Validation messages. From this icon, a panel is expanded presenting a list of messages explaining the violated rules;
- **Marking 5**: Execute and import buttons. The goals of this two buttons are, respectively, to generate and to import the generated widget, saving and installing it on the Target Platform.

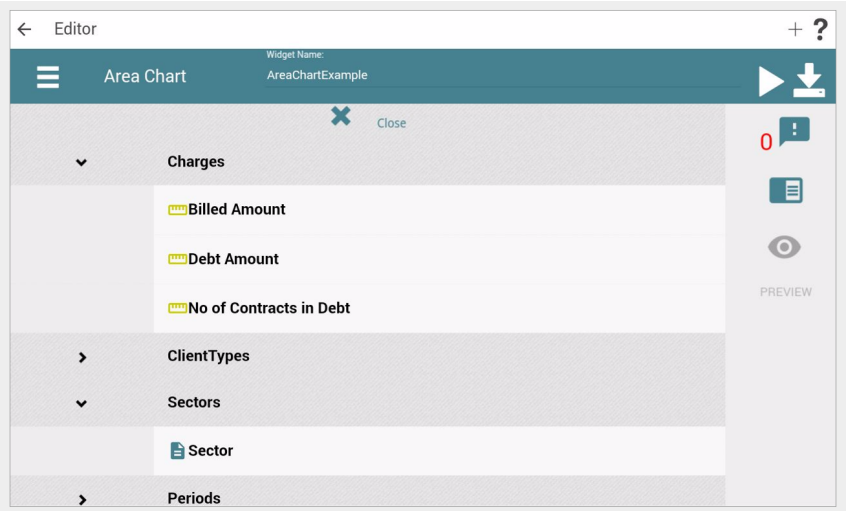


Figure 4.7: Meta-data list example. Version after usability tests

Let us suppose that the end user wants to configure a simple Area Chart showing the annual billed amount evolution, for each region. The end user starts by opening the Categories (X Axis) container and selects an attribute, namely the attribute Region. Then, the end user configures the variable to be displayed on the Y Axis and adds the year attribute on the Series. By adding the year on the Series, the information will be aggregated for each year and will illustrate it by each area, wherein each area is a year.

As illustrated on the figure 4.8, the attributes added to the containers are represented by the blocks. Each block contains its name and the symbol which indicates its type. This elements will be studied on the Modeling Language design (chapter 5).

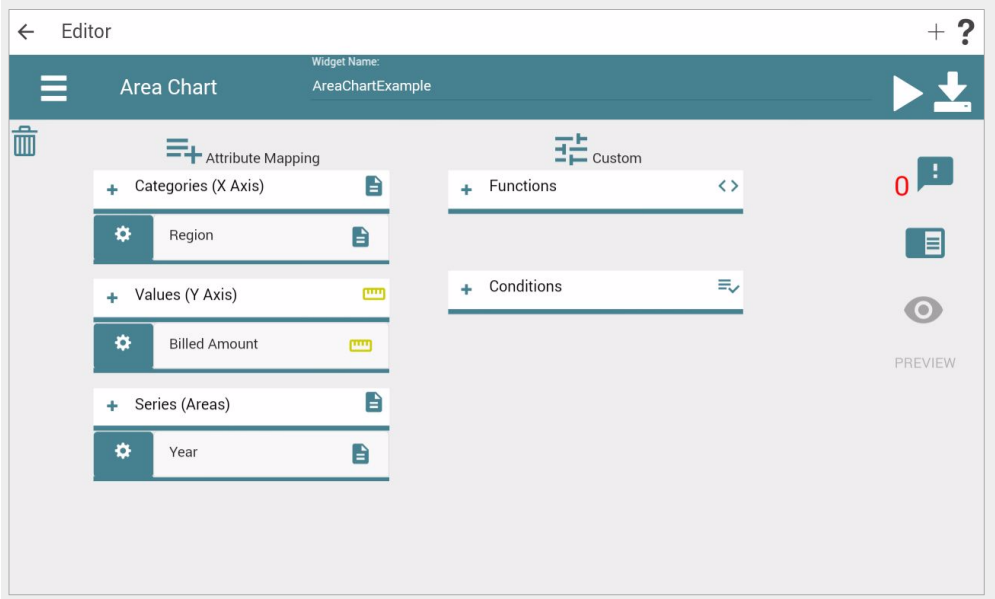


Figure 4.8: Containers populated with blocks

4.3.3 Generated Widgets UI specification

On the generated widget is presented a view fragment where is incorporated a set of elements.

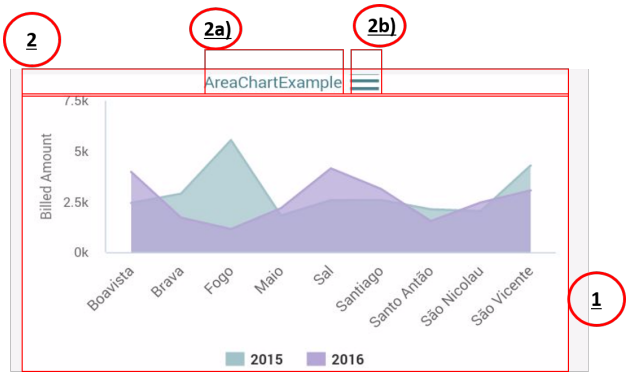


Figure 4.9: UI of a generated widget

Based on the markings of the figure 4.9, the following list describes each element:

- **Marking 1**: Widget body. It is the place where the main data-visualization component is drawn;
- **Marking 2**: Widget header. Composed by the widget title - **2a)** - and a menu button - **2b)**.

For further information, the figure on A.4 shows other examples of generated widgets.

4.3.4 Customization

Let us suppose that the end user had a question which cannot be fully answered by the widget generated and illustrated on the figure 4.9. The end user wanted an analysis where it was possible to analyse the billed amount by region, but at the same time, having a comparison regarding the current year versus its homologous information, so that the end user can take some conclusions related to either or not the company is making any progress.

For this specific case it is available a production asset named Homologous VS Current By Year (HVSC-Y) which the end user can pick to generate such widget. For this case the end user must model the same widget has the one illustrated on the figure 4.9, but in addition he must open the Functions container on the Editor and choose the HVSC-Y asset.

The generated result is represented on the figure 4.10. Based on its markings, the following list describes each element:

- **Marking 1**: Previous year data showing the homologous information;

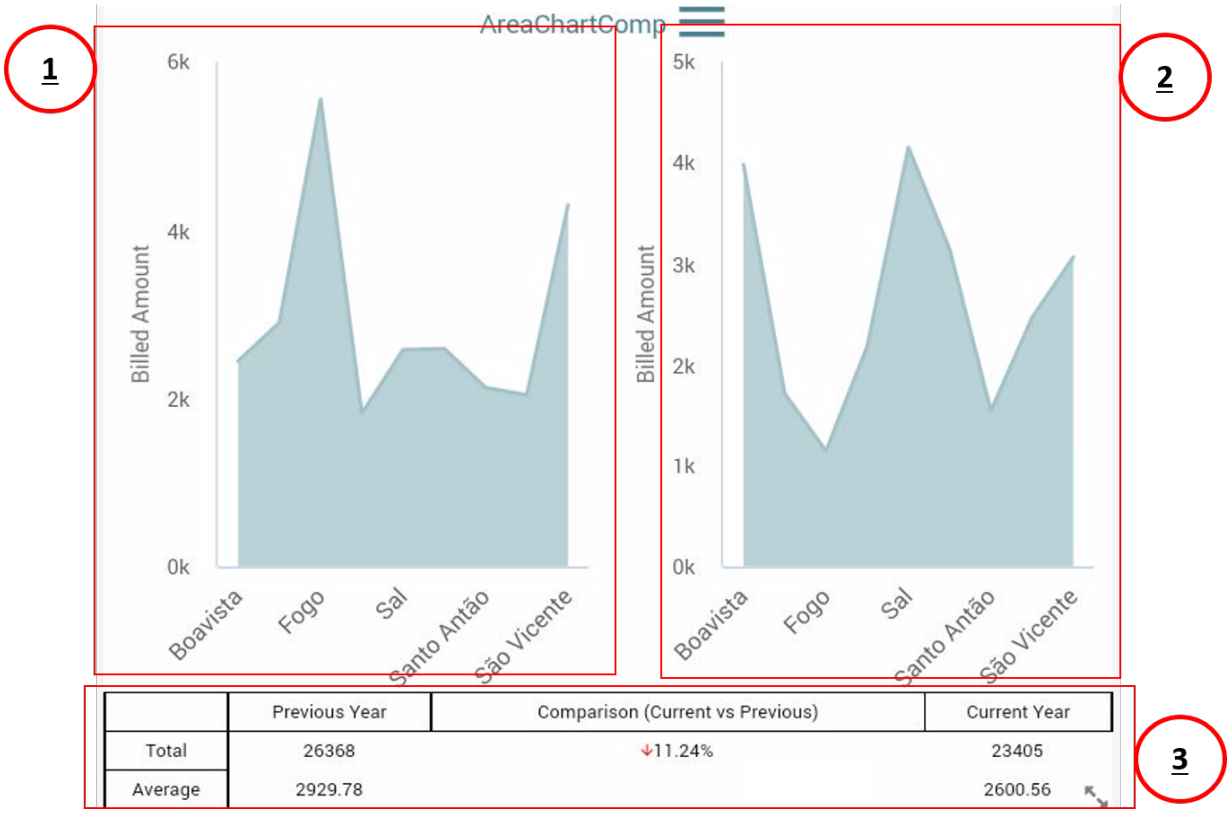


Figure 4.10: Homologous VS Current By Year: generated widget example

- **Marking 2**: Current year data;
- **Marking 3**: Summary data-table showing a billed amount comparison between the two years.

4.4 Summary and discussion

In this chapter we presented a case study related to the *Energy & Utilities* business area. In particular, we introduced some business related concepts as well as the analytical needs. We then introduced an overall solution and the traced plan to conceive it.

In the same chapter, we presented the user interface by introducing two front-end components along with some running examples related to the case study: the new version of the Target Platform and the new component denominated as Editor.

In general terms, the Editor is the component where the end user is empowered to configure its own analytical widgets. This is possible thanks to the domain-specific modeling language that resides in it, which is responsible for comprising the rules and syntax that enables the end user to specify the widgets in an abstract level - the model level. The model is then read by some component to generate the widget code which is compatible for execution on the Target Platform. This latter aspect forms a major part of the solution's core, which is yet to be explained.

Before we delve into the details of the solution's core implementation, the modeling language needs to be presented. In the next chapter resides the explanation related to the modeling language design, wherein it is covered all aspects regarding the syntax, its elements and rules managed by the modeling language.

5

The Modeling Language Design

In the previous chapter we introduced the overall solution. In this chapter the goal is to introduce some details related to the solution. Particularly, based on the concepts learned on the domain analysis - 2.1.5 -, this current chapter focuses on the Modeling Language - the main module used by the end user to configure the widgets. The concepts presented on the domain analysis are, therefore, relevant in order to understand this chapter.

Looking at the definition of Domain-Specific Language (DSL), fully explained on the chapter 2: it is a computer programming language that is oriented to a specific area of knowledge or activity, it concerns to a particular range of problems of a particular domain [Fow10]. Domain-specific Modeling Languages (DSML) maps this definition to the specification of models. Hence we can conclude that the designed Modeling Language is oriented to a specific domain and therefore is a Domain-Specific Modeling Language. As mentioned on the Domain Analysis chapter 2.1.5, in this specific work we have considered the Business Intelligence (BI) as the domain, focusing particularly on the concepts of multi-dimensional data modeling and the corporate data visualization. Once again, is important to remember that BI is an area which is broadly recognized by organizations as the support for decision making activities. It is a generic domain in the sense that its concepts are horizontal to business areas. Theoretically, this brings challenges and advantages because the modeling language should be valid, regardless of the business area context.

5.1 Language Design

Inspired by the concepts learned on the domain analysis, we established the following items as requirements for the Modeling Language:

- **REQ1:** It should capture the relevant concepts of Business Intelligence;
- **REQ2:** Also, it should present the concepts related with the business area;
- **REQ3:** Even though, it should be as generic as possible, upon the possibility of switching to different business areas. Leveraging the portability of the language concepts.

After the knowledge extracted from the domain analysis and established the aforementioned requirements, the language design was the next phase.

The first step of the language design is to define the Abstract Syntax. The Abstract Syntax describes the properties of the domain, defines the individual elements of the language and the rules that defines how those elements can be combined [Gre+04]. This is an important step in the language deployment as it is the base of the whole process.

The next step is to define the Concrete Syntax. The Concrete Syntax is the readable representation of the language and is what the user interacts with [Voe+13]. Therefore, the Concrete Syntax should be simple and represent the necessary and relevant concepts, in order to leverage the understandability.

Lastly, we defined a set of well-formedness rules to grant the validity of the model instantiated by the end user.

5.1.1 Designing the Abstract Syntax

As explained on 6.4.1, each widget has its own properties. Based on this idea, it is important to note that, alongside with the maturation of the whole system, the diversity of widget types is likely to rise. Therefore, the Abstract Syntax of the Modeling Language was modularized, in the sense that, each widget type has its own meta-model describing its specific properties, simplifying the deployment of new categories of widgets.

As an illustration regarding the differences between the widget types, the line chart and pie chart differs, between each other, on its specific properties. The line chart relies on Horizontal and Vertical Axis to display the data upon different Series. On the other hand, the pie chart relies on the Sectors and its Quantities to display the information.

At the same time, although the specific properties are different, the generic goal is common - to present the information. So, they are generally classified as Specific Properties, regardless of the widget type, mapping and displaying the data on the widget. This notion is conceptually represented on the model of the figure 5.1 - a Widget is classified by its Type and owns Data Mapping properties, each property being specific to the type of the Widget. Upon each Specific Property, the business information is mapped, as represented with the relationship with the Measure and Descriptive Blocks. The Measure Block represents the numerical values and the Descriptive Block the descriptive information to be presented by the widget. These are generalized as Data Blocks.

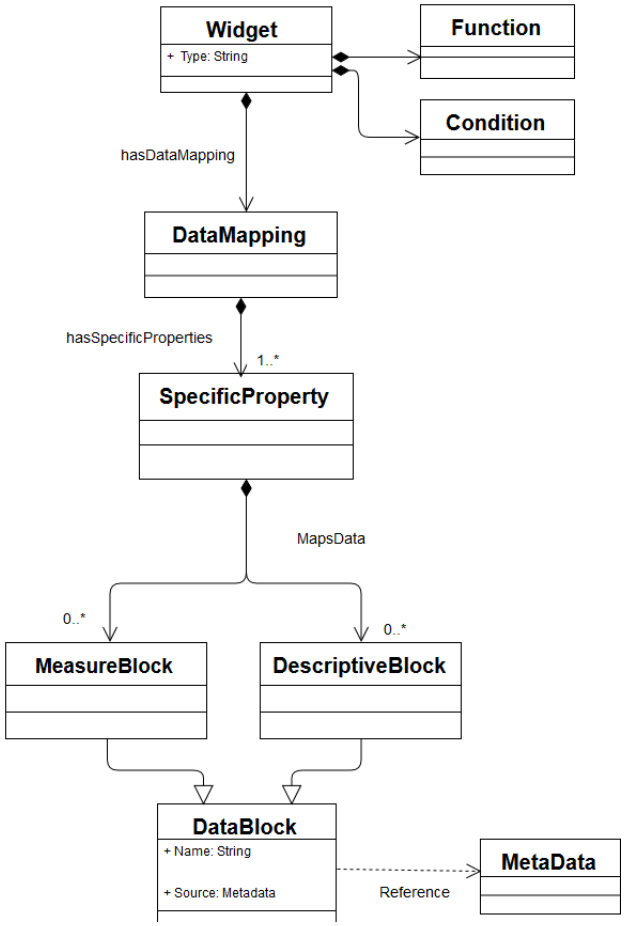


Figure 5.1: Abstract Syntax: Each widget has its specific Data Mapping properties

The Data Block is responsible for capturing the business concepts, by referencing its corporate data, as represented with relationship between the Data Block and the Meta-Data. Considering the definition of meta-data as data describing other data[1], the purpose of the class Meta-Data is to describe and capture the concepts of the corporate information, namely its data-model. This class is vertical to the business area concepts, meaning that, it needs to be re-configured on specific context switches, namely changing to other business areas.

This idea aligns with the stated requirements **REQ2** and **REQ3**, because, typically, the Meta-Data will be the only element subject to re-configurations on business related changes.

Lastly, and regardless of the widget type, the elements Condition and Function are designed to customize the widget and the data presentation.

In particular:

- The element Condition is responsible to represent the conditions over the Data Blocks, selecting the data to be presented, based on the verification of conditional

[1] http://www.oxforddictionaries.com/definition/english/metadata

expressions. This concept is useful in scenarios when the widget should only display the data that satisfies a specific constraint. Each condition instance is represented as a Condition Block. The latter aggregates one or more Data Blocks which participates on the conditional expression;

- The element Function, on the other hand, represents generic assets which can be used on the widget. These generic assets can have diverse goals, for example to customize and filter the data to be presented, to give the widget specific behaviors based on certain events, to customize the presentation of the widget. From the user perspective, this concept is useful in order to provide generic tools that can be directly used in the Editor to solve specific problems. Each generic asset instance is represented by a Function Block and aggregates one or more Data Blocks which are used to parametrize the asset.

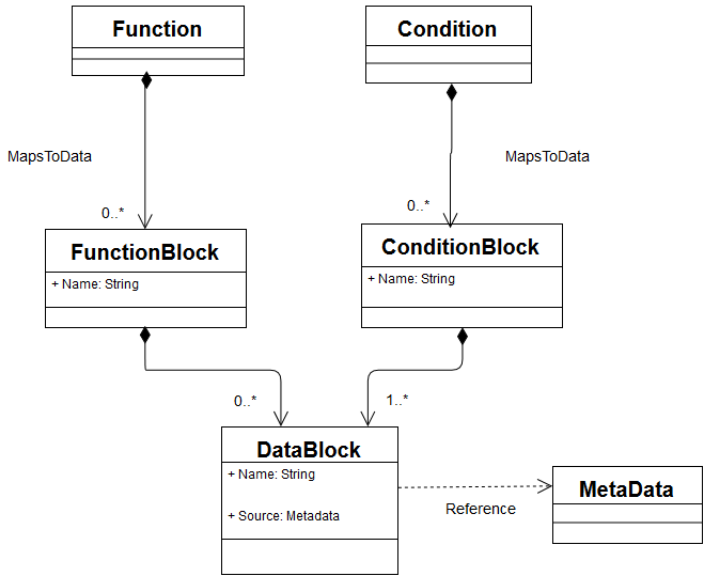


Figure 5.2: Abstract Syntax:The Function and Condition are common properties

The following Function assets were chose to be part of an initial deployment:

- **Sort Data:** The goal of this asset is to sort the data before presenting it on the widget. It accepts two parameters: *By* and *Mode*. The *By* parameter instructs the widget on which criteria it should sort the information. The *Mode* indicates how to sort, either ascending or descending;

- **Top N:** The goal of this asset is to present a classification showing the N best records. It accepts two parameters: *N* and *Ordered By*. The *N* parameter indicates the maximum records to show, the *Ordered By* instructs the widget on which criteria it should sort the information in order to present the best *N*;

- **Bottom N:** The goal of this asset is to present a classification showing the N worst records. It accepts two parameters: *N* and *Ordered By*. The Bottom N parameters

have the same goals as the Top N ones, the only difference resides in the sorting mode. In this case the data is sorted in ascending mode;

- **Homologous Versus Current By Year (HVSC-Y):** The goal of this asset is to extend the widget such that it presents an analysis regarding the current and the past year. On each analysis the End-User can visualize a specific aspect of the current year versus the homologous on the past year. This asset needs no parameter configuration, as it extracts, automatically the current year, that is, the last year instantiated on database, alongside with the previous one.

  - **HVSC-Y constraint:** To use this asset, the business context must have time-based dimensions which includes annually levels.

5.1.1.1 Abstract Syntax: Widgets Specific Properties

So far it was covered the generic concepts of the Modeling Language, we will then focus on the Specific Properties of each widget type.

The following type of widgets were chose to be part of an initial deployment of the Modeling Language: **Line Chart**; **Bar Chart**; **Area Chart**; **Pie Chart**; **Heat Map**; **Interactive Pivot Table**; **Drop Down Filter**. The definition of each one can be found on the Domain Analysis (chapter 2.1.5).

The simple **Line Chart**, **Bar Chart** and **Area Chart** relies on the following properties:

- Y Axis: Vertical axis of the chart. It maps and displays the numeric data;
- X Axis: Horizontal axis of the chart. It maps and displays the descriptive information;
- Series: Lines/Areas/Bars of the chart. It maps and displays the descriptive information over different series.

These concepts are conceptually represented on the diagram of the figure 5.7. The YAxis element is, uniquely, composed by Measure Blocks and the XAxis and Series are composed by Descriptive Blocks.

For the **Pie Chart**, the specific data mapping properties are:

- Sectors: Slices of the pie. It maps and displays the descriptive information;
- Quantity: It maps and displays the numeric data for each Sector;

The **Heatmap** relies on the following properties:

- Rows: Rows of the matrix. It maps and displays the descriptive information;
- Columns: Columns of the matrix. It maps and displays the descriptive information;
- Cells: Cells of the matrix. It maps and displays the numeric data, for each row and column coordinate.

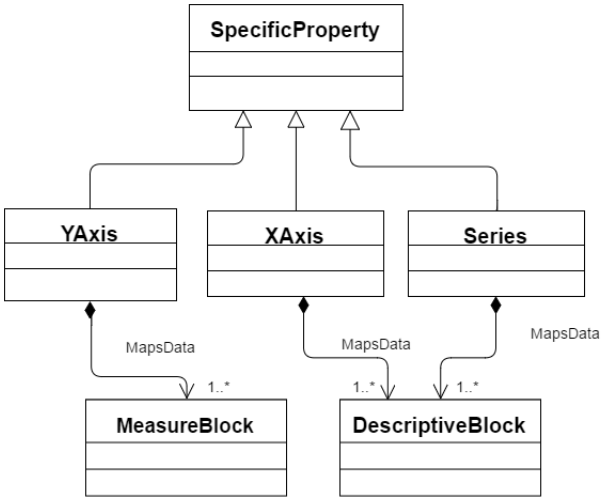


Figure 5.3: Abstract Syntax:Properties of the Line, Bar and Area chart. For sake of simplicity the other elements were hidden, namely the Data Block and Meta-Data

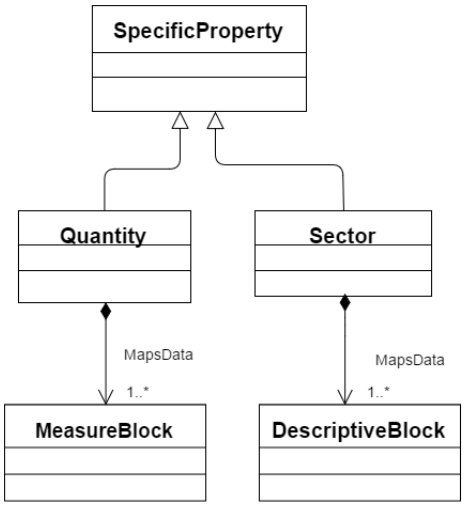


Figure 5.4: Abstract Syntax:Properties of the Pie Chart

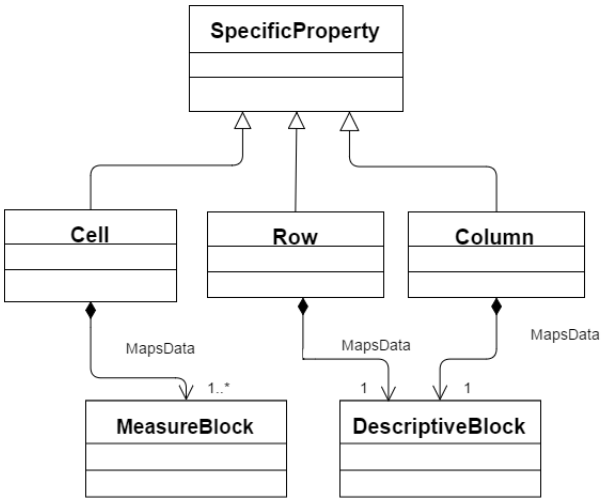


Figure 5.5: Abstract Syntax:Properties of the Heat Map

The **Interactive Pivot Table** relies, uniquely, on the Attribute property. The Attribute represents the Data Blocks that are mapped, forming a multi-dimensional cube that is fed to the Pivot Table.

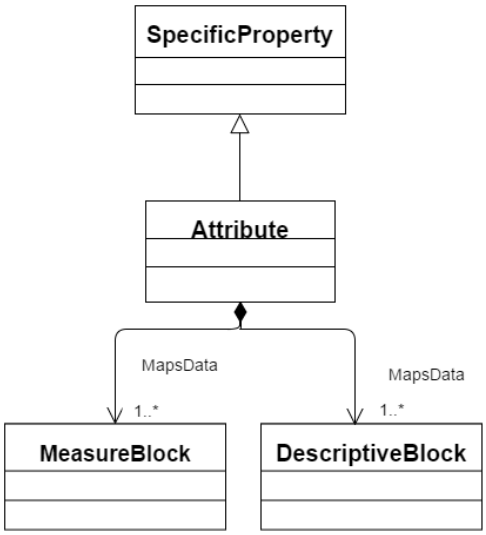


Figure 5.6: Abstract Syntax:Properties of the Pivot Table

The **Drop Down Filter** also relies, uniquely, on the Drop Down Value property. The Drop Down Value represents the descriptive Data Blocks that are mapped, forming a list of descriptive and distinct data for selection by the user. The selected data performs a slice operation upon the information presented on the other widgets.

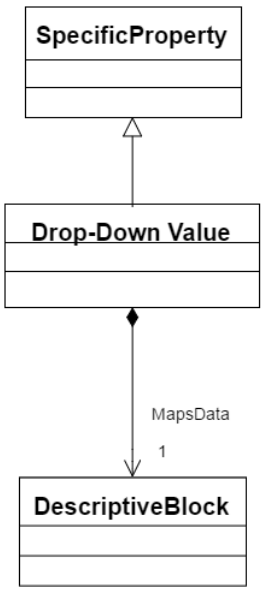


Figure 5.7: Abstract Syntax:Properties of the Drop Down Filter

5.1.1.2 Abstract Syntax: Structuring the Meta-Data

The Meta-Data is a structure which captures the business related concepts based on the data-model. As described on the Domain Analysis, the data-model is an abstract multi-dimensional representation of the information containing attributes that are classified as measures or descriptors. One of the implementation methods of the multi-dimensional model is, typically, based on the design of a relational database composed by fact tables and dimension tables.

The Meta-Data used by the Modeling Language is based upon multi-dimensional models on star-schema formats and implemented with relational databases. Therefore, the Meta-Data is, internally, composed by elements that describes the fact-tables and its measures, the dimensional entities and its descriptive attributes, alongside with elements that describes the relationships between these entities. This notion is represented on the diagram of the figure 5.8.

The class Meta-Data is composed by the relational entities, Fact-Table and Dimension-Table. The latter is composed by the class Level, this class represents the descriptive attributes and is described by:

- *Name*: Name of the attribute, it is the same as the one configured on database;
- *Alias*: Custom name of the attribute. Typically different from the name configured on database. It is useful in order to the Editor present a more user-friendly name;
- *LevelID*: Unique identifier name. This attribute represents the identifier (ID) of the hierarchy-level, if there is embedded hierarchies on the Dimension Table. If not, then it is simply an ID identifies the records, used to join with the fact table.

The Fact-Table is composed by its measures, represented by the class Measure, and the foreign-keys, represented by the class Foreign-Key. The Measure is described by its Name and Alias, each one with same goal of the respective ones on the Level Class. The class Foreign-Key is described by the following attributes:

- *Name*: Name of the foreign key attribute, is the same as the configured on database;
- *PointsTo*: Reference to the Dimension Table;
- *Level*: Reference to which level of the Dimension it connects to; If the Dimension has no embedded hierarchies, then it simply points to the record ID.

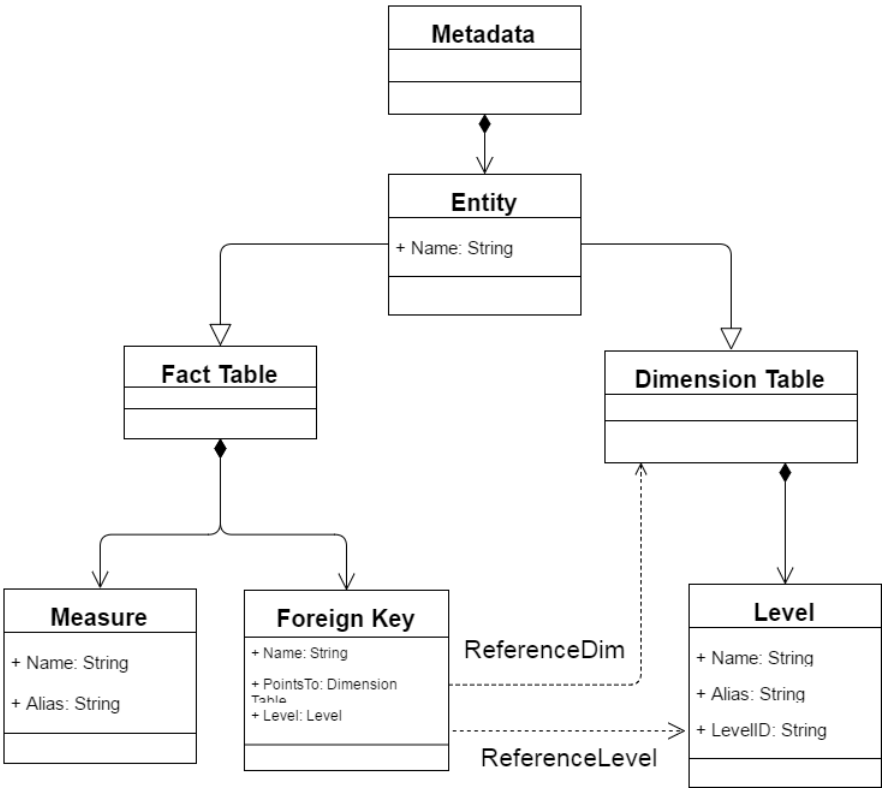


Figure 5.8: Abstract Syntax:Meta-Data

5.1.2 Designing the Concrete Syntax

The Concrete Syntax design is an important step in order to have an understandable and well structured language. As aforementioned, the concepts of the Concrete Syntax are directly related with the front-end interface, and is what the user interacts with. It maps the concepts of the Abstract Syntax to the visual elements of the Modeling Language.

The mapped concepts from the Abstract Syntax are the block elements, the specific and common properties of the widgets. Each block is depicted and described on the table 5.1. The specific and common properties are illustrated and described on the table 5.2.

Table 5.1: Blocks concrete syntax

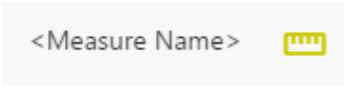
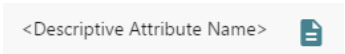
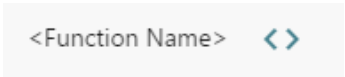
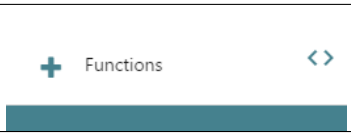
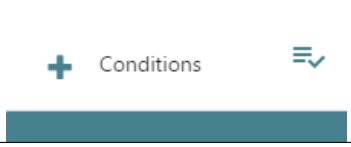
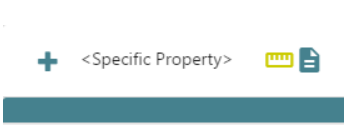
Graphical Element	*Meta-model Element*	*Description*
	Measure Block	Represents the business measures. Its icon was adopted, inspired by existing tools well-know on BI area, namely from the SAP WEBI
	Descriptive Block	Represents the business contextual attributes
	Function Block	Generic assets used to customize the widget or the presented data.
	Condition Block	Conditional expressions to filter the presented information.

Table 5.2: Concrete syntax regarding the common and specific properties. The icon on the top right corner, indicates the type of data-blocks that each container accepts

Graphical Element	*Meta-model Element*	*Description*
	Function	Container of the generic assets. Each Function Block added in this container represents an asset instance.
	Condition	Container of the conditions. Each Condition Block added in this container represents a condition instance.
	Specific Property	This generic container represents a specific property. For each property of the widget, this container is instantiated. Each Data Block added on it, represents the information to be presented on the property of the widget.

5.1.3 Well-Formedness rules

To ensure every model instantiated by the end user is valid, some well-formedness rules were defined.

5.1.3.1 Rules regarding the widget specific properties

The **line**, **bar** and **area charts** rests upon the following rules:

- *Y Axis is defined:* The meta-data attributes must be mapped on this property, in order to present the ordinate values;
- *X Axis is defined:* The meta-data attributes must be mapped on this property, in order to chart present the abscissa values.

The configuration of the **pie chart** relies on the following rules:

- *Sector is defined:* The meta-data attributes must be mapped on this property, in order to present the categories which defines the slices of the pie;
- *Quantity is defined:* The meta-data attributes must be mapped on this property, in order to define which quantities are represented by the sectors.

For the **heat map**, the rules are:

- *Cell is defined:* The meta-data attributes must be mapped on this property, in order to define which numeric values will be contained on the cells;
- *Row is defined:* The meta-data attributes must be mapped on this property, in order to define which descriptive values will be contained on the rows;
- *Column is defined:* Same purpose as the rows, but instead, the rule is applied on the columns.

For the **pivot table**, the rules are:

- *Attribute is defined:* The meta-data attributes must be mapped on this property, in order to define which attributes will form the pivot-table data-set.

For the **drop down filter** , the rules are:

- *Drop Down Attribute is defined:* The meta-data attributes must be mapped on this property, in order to define which attribute will form the list-of-values of the drop-down filter.

5.1.3.2 Common rules

- *Top N and Bottom N parameters are always defined:* The parameter *N* and *Ordered By* must be defined on these two assets, in order to define the number of records to display and the criteria of classification;
- *Sort Data parameters are always defined:* The parameter *By* and *Mode* must be defined in order to define on which attribute and how the data should be ordered;
- *Top N, Bottom N and Sort Data are mutually exclusive:* Since the goal of the Top N and the Bottom N are opposite, this rule prevents the simultaneous selection of both assets;
- *Top N, Bottom N or HVSC-Y are unique:* It is not possible to select two or more instances, for each one of these assets;
- *Avoiding Sort Data parametrization conflicts:* It must be avoided the cases where the user configures two or more Sort Data assets, wherein exists parametrization conflicts. In particular, the scenarios where two or more assets sorts the data with the same *By* parameter value, nonetheless, with different *Mode* values.

5.2 Summary

In this chapter we have presented the design of the Modeling Language. As aforementioned, the language design started by designing the Abstract Syntax, it enabled us to identify conceptual properties which were important to conceive the Concrete Syntax. The Abstract Syntax was modularized, so that, each widget type has its own metamodel. This way, the system can cope with further deployments regarding different widget types.

The Concrete Syntax was the next phase, here we followed an approach were the elements of the language are mainly represented as blocks and containers. The blocks are the ones which contains the business related concepts and are mapped on the properties of the widgets, represented as containers.

Lastly, we defined some well-formedness rules with the goal of verify some invariants regarding the models instantiated by the end user.

In the next chapter we will delve further into the solution's core, by explaining how this concepts were implemented and how the necessary components were defined.

6

Implementation

Once again, the main objective is to empower the end user on the configuration and generation of analytical widgets, based on a cooperative strategy along with the domain experts. One important module is the interface which the end user interacts with, consisted by the Modeling Language that increases the abstraction level. So far, it was presented the language design. From here, it is important to implement the facilities which will support the language design concepts and cover the objectives initially identified. The final result will be an infrastructure which implements the theoretical concepts of MDD and aligns with the principles of SPL.

The development passed by first implementing the abstract syntax concepts from the Model Language Design, by implementing the widget meta-models and the meta-data. Simultaneously, the architecture was designed, by identifying the main infrastructures and the strategy of integration. Finally, the implementation took place with the construction and integration of each component.

As such, we have designed and built a modular system composed by an editor (Editor Component), a server (Server Component) and a mobile application (Mobile Component or Target Platform). In a nutshell, the Server is responsible to implement the MDD concepts by managing the transformation process which reads the configuration made on the Editor. The result of such process is interpreted by the Target Platform.

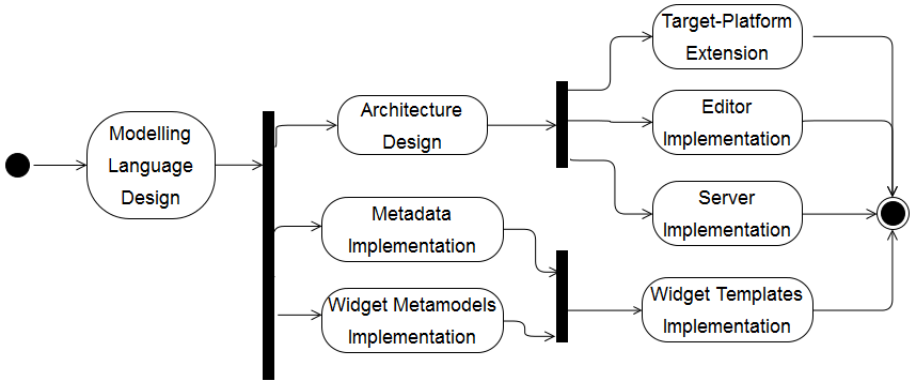


Figure 6.1: Implemention Stages

6.1 Meta-models and meta-data implementation

The widget meta-models and the meta-data are the input of the Editor Component. They represent, in a textual notation, the abstract concepts described on the Modeling Language Design. In particular, these assets were implemented on Javascript Object Notation[1] (JSON), serialized on JSON extension files and deployed on the Server.

6.1.1 Line, Bar and Area Chart

Each property of the line, area and bar chart is described by the JSON object illustrated on the listing 6.1. In particular, the attribute AllowedType is a constraint that describes, for each property, the type of Data Blocks that are accepted.

Listing 6.1: JSON meta-model example describing the properties of the line bar and area chart

```
1  {
2      Name: "Line Chart",
3      DataMapping : [
4          {Name:"XAxis", AllowedType:"DescriptiveBlock"},
5          {Name:"YAxis", AllowedType:"MeasureBlock"},
6          {Name:"Series", AllowedType:"DescriptiveBlock"}
7      ]
8  }
```

6.1.2 Pie Chart

Listing 6.2: JSON meta-model example describing the properties of the pie chart

```
1  {
2      Name: "Pie Chart",
3      DataMapping : [
4          {Name:"Sectors", AllowedType:"DescriptiveBlock"},
```

[1] http://www.json.org/

```
            {Name:"Quantity", AllowedType:"MeasureBlock"}
        ]
    }
```

6.1.3 Heat Map

The heat map counts with one more specific constraint. For each property, the MaxValue limits the number of Data Blocks that are possible to map.

Listing 6.3: JSON meta-model example describing the properties of the heat map

```
{
    Name: "Heat Map",
    DataMapping : [
        {Name:"Row", AllowedType:"DescriptiveBlock", MaxValues:1},
        {Name:"Column", AllowedType:"DescriptiveBlock", MaxValues:1},
        {Name:"Cell", AllowedType:"MeasureBlock"}
    ]
}
```

6.1.4 Pivot Table

The pivot table counts only with the Attribute property, which accepts measures or descriptive blocks.

Listing 6.4: JSON Meta-model example describing the properties of the pivot table

```
{
    Name: "Pivot Table",
    DataMapping : [
    {
        Name:"Attributes",

        AllowedType:["DescriptiveBlock","MeasureBlock"]
    }
    ]
}
```

6.1.5 Drop down filter

The drop down filter counts with the DropdownValues property, which accepts only one descriptive block.

Listing 6.5: Meta-model example describing the properties of the drop down filter

```
{
    Name: "Drop Down",
    DataMapping : [
        {Name:"DropdownValues",AllowedType:"DescriptiveBlock",MaxValues:1}
    ]}
```

6.1.6 Meta-data

The Meta-Data file contains objects identifying and describing each entity from the relational database. For the fact tables, the Meta-Data file contains the description about the measures and foreign keys. For the dimension tables, the Meta-Data contains the information about the descriptive attributes. Every aspect implemented on the Meta-Data is conformed with the design conceived and studied on the Modeling Language design chapter (chapter 5).

The example given on the listing 6.6, presents a Meta-Data implementation where is configured one fact table and one dimension table - the *charges* and the *periods*, respectively.

The *charges* meta-data contains two elements - the *fks* and the *measures*. According to Meta-Data design, on the the Modeling Language Design chapter (section 5.1.1.2), a fact table is composed by measures and the foreign keys, which references the dimension entities. Therefore, the goal of the *fks* is to identify and describe the foreign keys. In the same example, one foreign key is identified - the *period_id* where it is described its name and the reference to the dimension table. The reference to the dimension table is granted by two attributes - the *pointsTo* and the *levelId*. This information will be useful when the code-generation process takes place, in particular, to generate the SQL query that will retrieve the data to be presented by the widget. The *measures* attribute objective is to identify and describe the measures contained by the fact table. This last element, in particular, will be useful in order to the Editor list the measures contained on the multi-dimensional model.

The *periods* meta-data example contains one element - the *levels*. Each of the elements contained on the *levels* are, essentially, the attributes of the dimension, where, in this particular example, it is configured the month and the year attribute. These elements will be useful for the editor list the descriptive attributes contained on the multi-dimensional model.

Listing 6.6: Meta-data excerpt describing the case study multi-dimensional model

```
1   {
2       FactTables: [
3       {
4           id: "f0"
5         name:"charges,"
6         fks: [
7           {name:"period_id",pointsTo:"d3",levelId:"0"}
8         ],
9         measures : [
10          {name:"billed_value_currency",alias:"BilledValue",source:"f0"},
11          {name:"debt_value_currency",alias:"ValueInDebt",source:"f0"}
12           ]
13      }],
14      DimensionTables: [
15         {
```

```
16            name:"periods",
17            id:"d3",
18            levels:[
19              {id:"0",name:"month",levelId:"id",alias:"Mes",nodes:[]},
20              {id:"1",name:"year",levelId:"id",alias:"Ano",nodes:[]}
21                ]}
22          ]}
```

6.2 Model instance

The widget model is an abstraction layer that describes what information is presented by each property. It is instantiated by mapping the meta-data attributes to the widget properties. This action is made by the end user, when modelling the widget on the Editor.

The listing 6.7 gives an example of a line-chart model, the end user configured the *month* on the X axis and the measure *value_currency* as the measure to be presented on the Y axis. Hence, the model describes a line chart in which, is presented a time-based evolution, by month, of the billing.

Listing 6.7: Model instance describing a line-chart

```
1  {
2     Name: "Line Chart",
3     WidgetName: "BillingPerMonth",
4     DataMapping : [
5     {Name:XAxis, Values:[{name:"month",levelId:"id",alias:"Month",nodes:[]}]},
6
7     {Name:YAxis,
8     Values:[{name:"billed_value_currency",
9     alias:"Billed Amount(Euros)",source:"f0"}]},
10
11    {Name:Series,Values:[]}
12    ]
13 }
```

This model instance will be inspected by validation and transformation processes, on the Server Component, to grant the well-formed rules and to generate the code executable on the Target Platform.

The next sections (6.3 and 6.4) will be dedicated to explain how these components were conceived.

6.3 Architecture design

The figure 6.2 depicts the final architecture, with the technologies instantiated, the goals of each one will be described throughout this chapter.

The Server Component is the kernel of the system. It is responsible to manage the deployed assets, for providing code-generation and model-validation services. Additionally, it is responsible to provide access to the Editor, via RESTful services, in which is provided the Modeling Language.

The Mobile Component, also denominated as the Target Platform, in addition to incorporate and present the Editor within an Iframe element, is also responsible for importing the generated widgets from the Server and save it on the local Widget Repository. Then, the imported widgets are accessible to the end user, to configure the dashboards.

This architectural design introduces the following advantages:

- **The Editor is not tightly coupled to the Target Platform device:** Because this component is accessed via REST requests, it can be used with web-browsers in non-portable devices, opening the opportunity to model the widgets even in back-office contexts. Additionally, it can be integrated in many other similar web-based projects of the company;

- **The Target Platform is agnostic to changes made upon the Editor:** The changes made by the developers on the Editor are reflected on the user device, on page refreshing and cache cleaning. Thus, it is possible to change the Editor without re-building the Target Platform and vice-versa;

- **The dashboard can be configured even in offline contexts:** Since the generated widgets are imported to the Target Platform and the summarized corporate data resides in its local database, it will be possible to edit the dashboard presentation on the mobile device, by configuring the widgets disposal, even in offline situations.

Nonetheless, it introduces the following challenges:

- **The usability is constrained on the services reachability:** Because the services are centralized and the Editor is accessed remotely via an Iframe, it is impossible, in offline or low-signal situations, to create new widgets anywhere and anytime;

- **The usability is constrained on the services availability:** The capability of generating widgets, is also constrained on the capacity of the services being always running and available for communication with the mobile device.

In the section 6.4, is described, in detail, every architecture module, in terms of implementation.

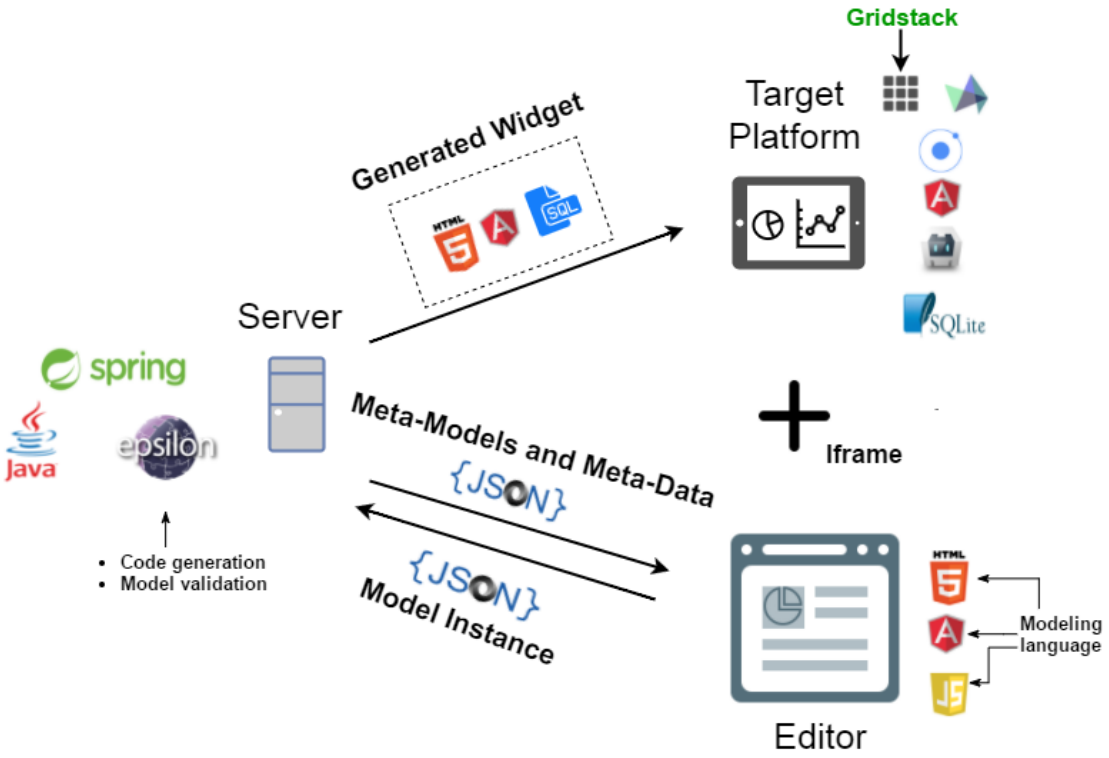


Figure 6.2: Designed architecture instantiated with the technologies

6.4 Architecture implementation

6.4.1 Editor Component

- **Component Input:** Meta-Models and Meta-Data;
- **Component Output:** Model instance;

The features offered by the Target Platform, already gives to the end user, the capability of visualize and interact with dashboards. Nonetheless, this empowerment level is not enough if the user needs to construct new widgets and configure the dashboard presentation. For this reason, the Editor Component was conceived. In particular, to empower the end user on the widget construction.

From a technological point-of-view, the Editor is implemented with web-application standard technologies like HTML5, CSS3, Javascript and with the AngularJS (Version one) framework [2]. It is deployed on the Server and accessible on the Target Platform via Iframe.

The main goal of this architectural component is to empower the end user by offering a Modeling Language for widget construction and by establishing a mean of interface between the end user and the core-services. As aforementioned, the Modeling Language is based upon the concept of configuring which information is to be presented on each

[2] https://angularjs.org/

specific property of the widget. In this sense, the Editor should be capable of adapt its content based on which widget type is selected.

For this reason, it was conceived as a SPA (Single-Page Application), in the sense that, the interface of the Editor is dynamic, updating and adapting its presentation based on which widget template is selected. This way, it means that, the Editor page is requested to the server only once, and is dynamically adapted based on which resource is selected, without requesting the server for further updated pages or view fragments.

To realize this idea, as described before, each widget type has its own meta-model file, deployed on the server and serialized in JSON notation, which describes the specific properties and constraints. Meaning that, when one widget type is selected by the end user, its meta-model resource is interpreted by the Editor and, subsequently, the presented content is updated in order to display the properties.

In order to provide this features, the Editor was implemented based on the Angular core features and patterns. In particular, the front-end is composed by a set of modules, where, each one is dynamically compiled and rendered based on the widget meta-model and updated on the actions made by the end user. The modules are denominated as containers or shelves, which are, essentially, customized HTML elements and directives that are based on core directives of Angular JS, namely the *ngRepeat* [3] and the *ngBind* [4].

As generally represented on the listing 6.8, given the widget template properties, the containers are displayed on the Editor, basing upon the *ngRepeat* to allocate each container and on the *ngBind* to bind and show the specific information of each one - for example its name. Additionally, each container accepts blocks representing the information to be presented on the widget. The blocks added on the containers and every other action made by end user, is projected on the model instance, using the data-binding features of Angular JS.

Listing 6.8: Editor HTML Document and Angular ngRepeat and ngBind Directives

```
1 <container ng-repeat="property␣in␣metamodel.DataMapping" ng-bind="property.Name">
2 <block ng-repeat="block␣in␣property.Values" ng-bind="block.name"></block>
3 </container>
```

Taking the pie chart as a brief illustration, Highcharts [5] defines it as a circular chart that is divided in **sectors** which is proportional to the **quantity** it represents. A meta-model of a Pie-Chart is, therefore, a file that adapts the Editor presentation based on the properties of the pie chart, as showed on the listing 6.2. Thus, the Editor, on this case, would present the containers to configure the Sectors and the Quantity, as depicted on the figure 6.3.

On the same example, based on the end user interactivity, upon the Sector and Quantity containers, were added, respectively, blocks representing the information about the Products and its Clients in Debt quantity. These interactions over the Editor interface

[3] https://docs.angularjs.org/api/ng/directive/ngRepeat
[4] https://docs.angularjs.org/api/ng/directive/ngBind
[5] http://www.highcharts.com/docs/chart-and-series-types/pie-chart

generates the serialized JSON model instance that describes the widget and the configurations made by the user - like the one illustrated on the listing 6.7. On submitting event, the model is sent to the server for validation and code-generation.

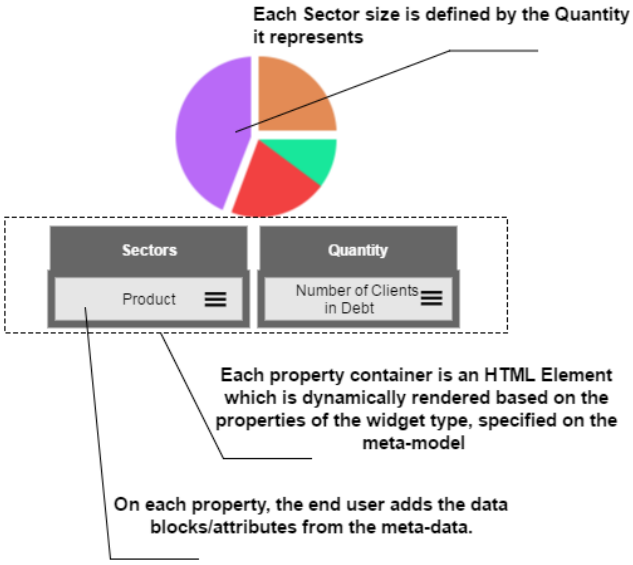


Figure 6.3: Pie chart basic example: The Editor renders the containers for the Sectors and Quantity properties based on the meta-model

6.4.2 Server Component

- **Component Input:** Model instance;
- **Component Output:** Generated code or a list of validation messages;

Although the model generated on the Editor is the result of a major step on the widget generation process, it is not sufficient to be interpreted by the Target Platform, given that it is only an abstract representation of the widget. Hence, it was necessary a component capable of interpret the model and generate the widget code, which is compatible with the Target Platform.

The Server Component was implemented with the purpose of respond the aforementioned challenge. It is the system kernel, being responsible for managing the deployed assets and by orchestrating the internal services, namely the model-validation and the code-generation. The services are based upon the Epsilon Task Specific Languages and its APIs and accessible to external entities via REST requests.

In this sense, the server is constituted by two main services:

- **Core services:** Model Validation and Model-To-Code Generation;
- **Web services:** RESTFul web service;

The Model Validation is based upon the Epsilon Validation Language (EVL) features and concrete syntax. It is responsible to ensure the well-formed rules, by validating the

models instantiated by the end users on the Editor. The Model-To-Code Generation is based on the Epsilon Generation Language (EGL) features and concrete syntax. It is responsible to consume the instantiated model and to read the template in order to generate the widget code. The code-generation is the actual transformation process explained on the Theoretical Concepts chapter (subsection 2.1.2.4), which defines the core of the Model-Driven concept. The RESTFul web service is the dispatcher, by mapping URIs to the core services.

6.4.2.1 Core services integration

Before the explanation regarding the core services implementation, let us identify and explain some of the the decisions that were taken.

As documented on the framework web-page [6], the Epsilon is strongly supported by the Eclipse Modelling Framework (EMF) plus the Graphical Modelling Framework (GMF) for the front-end and the Eclipse IDE [7] as the support. This unified technological stack grants little effort for the developer on complex integration tasks, besides installing the necessary plug-ins on the Eclipse IDE and some configurations related with the GMF editor.

For example, the presentation of model-validation results. If this process generates an error or warning, its messages are directly pointed to the inconsistent model elements in the GMF editor, describing the details that contributed for the inconsistency, alongside with an wizard based interface that supports the end user to resolve the issue. The configuration of such feature takes little effort from the developer.

In context of this work, adopting this stack would introduce a bottleneck in terms of accessibility and usability, as the GMF is neither aimed or optimal for user interaction on mobile devices. Thus, in the cases that, the end user wants to construct new widgets while on mobile context, it would be necessary to access the GMF editor via back-office, on desktop devices. This obstacle re-enforced our idea that integrating the GMF on the architecture, would not be the best solution and, therefore, a web-based editor would suffice to resolve this bottleneck.

At this point, the decision was to implement the web-based editor that despite of not being based upon the GMF, it would be easily integrated on mobile devices, leading to the following challenges on the server-side:

- When using the Epsilon, EMF and GMF as technology stack, the model-validation and model-to-code generation features are automatically executed on GMF runtime, when requested by the end user. Not using this technological stack, means that the EGL and EVL files needs to be executed as standalone processes. Hence, the challenge resides in how to add these modules on the server context, so that they can be loaded and executed on remote requests made by the end user.

[6] http://www.eclipse.org/epsilon/
[7] https://eclipse.org/

In order to realize this idea, we needed to implement Java-based standalone modules, which, when called on remote requests, it relies entirely on the Epsilon APIs to load the received model instances, load and execute the EVL, EGL files for validation and code-generation. This aspect is illustrated on the figure 6.4;

- Another challenge is related to the model technology/notation to use. The standard technological stack uses the EMF technology to manage and XMI to serialize the models. However, as the editor and the mobile-component are web-based applications, the preferable notation would be the Javascript Object Notation (JSON).

  As such, in order to integrate JSON with the task specific languages of Epsilon, we needed to incorporate an adapter on the Epsilon Model Connectivity (EMC). Epsilon documentation [8] defines EMC as an open model connectivity framework which developers can extend with support for additional types of models/modeling technologies by providing respective drivers. After some searching, a JSON driver for EMC was found [9]. It is an open-source driver that implements an API connecting JSON models to the internal structures of Epsilon. As a result, the JSON models can be accessed by the Epsilon Task Specific Languages for validation and code-generation.

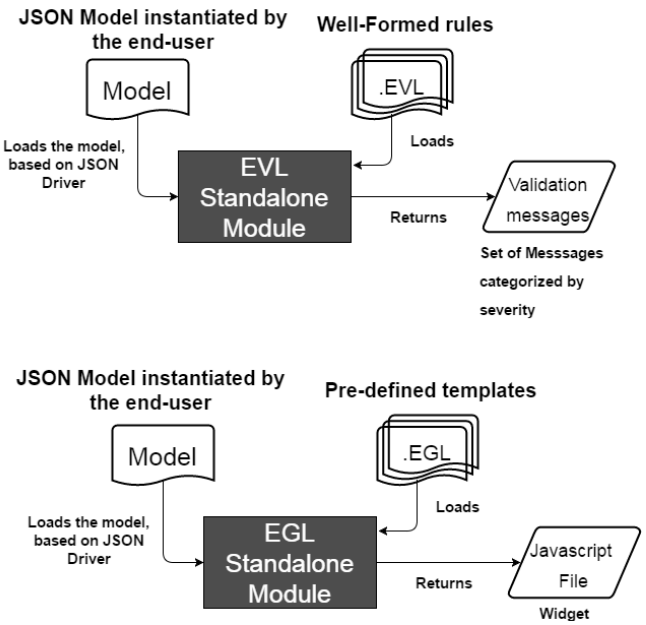


Figure 6.4: Standalone Modules: EVL Standalone Module loads the user instantiated model and the well-formed rules files for Model-Validation. EGL Standalone Module loads the user model and the widget template for Model-To-Code-Generation. Both uses the JSON Driver to integrate the model instance with the Epsilon features.

[8] http://www.eclipse.org/epsilon/doc/
[9] https://github.com/epsilonlabs/emc-json

6.4.2.2 Model Validation

The well-formed rules are the input for this service. Physically, they are represented by EVL extension files that contains the invariants for each widget type. Each widget type has its own EVL file, executed by the EVL Standalone Module, before the code-generation process.

Each invariant is categorized in two types: *critique* and *constraint*. This types belongs to the concrete syntax of EVL and if one of them is violated, it is returned a set of validation messages.

The validation outcome is classified on its severity level - warning or error. The warning is generated when a *critique* rule is violated, an error is generated when a *constraint* rules is violated. Depending on the result of the validation, the code-generation process is triggered. A validation which returns a warning message enables the code-generation execution, being useful to give support and advice or to alert the user to any possible inconsistencies that, despite of not compromising the widget presentation or processing, it may generate unexpected results. On the other hand, an error message result does not trigger the code-generation process, being useful in scenarios in which the mistakes made by the end user compromises the widget generation, processing or presentation. The listing 6.9 provides an example of a constraint rule.

Listing 6.9: Well-formed rule excerpt - EVL Syntax. On the *check* clause the EVL rule is specified. On the *Message* clause a customized message is written to be presented to the end user when the rule is violated

```
1 constraint yAxisHasValues {
2     check: DataMapping.selectOne(t|t.Name="YAxis").Values.size() > 0
3     message: "Y␣Axis␣shelve␣should␣have␣attributes␣in␣order␣to␣display
4 the␣values␣on␣the␣axis"
5 }
```

6.4.2.3 Model-To-Code Generation

The EGL files plays an important role on this service. It contains the template code of the widget, which is executed and matched with the received model, to generate the code.

Compatibility is one important aspect, as the Target Platform and the generated code must comply with each other such that the widget can be presented. As previously described on the sub-section 2.2.2.2, the Target Platform is a web application, developed with HTML, Angular JS and SQL Lite for data management. A generated widget has the same characteristics, in the sense that, the core is implemented with the same technological stack. Thus, it depends on HTML for document structuring, Angular JS features and SQL for data-retrieval. Physically, each widget is a modular piece, represented as a Javascript extension file that is saved on the mobile device storage, lazily loaded and executed by the Target Platform. The code-generation depends on this characteristics, to

generate the physical widget file. Hence, the widget structure was defined, such that the EGL template complies with this aspects.

The figure 6.5 illustrates the widget internal structure relationships. Each sub-component represents a set of features which are responsible to extract the information, map it on the properties and to draw the widget. They are triggered by the Target-Platform, by accessing the widget's generated API.

The execution of these internal features is, essentially, a data-flow process which leads to the widget drawing. The Data Extraction, returns the data stored on the mobile device by querying the local database. The data returned by the database is, then, managed by the Data Mapping features to map the information on the widget properties. It depends on the Data Extraction to execute its goal, as it cannot display the information without first extracting it. This aspect is controlled by the Manager, which manages the process control-flow. The overall pseudo-template is specified on the listing 6.12.

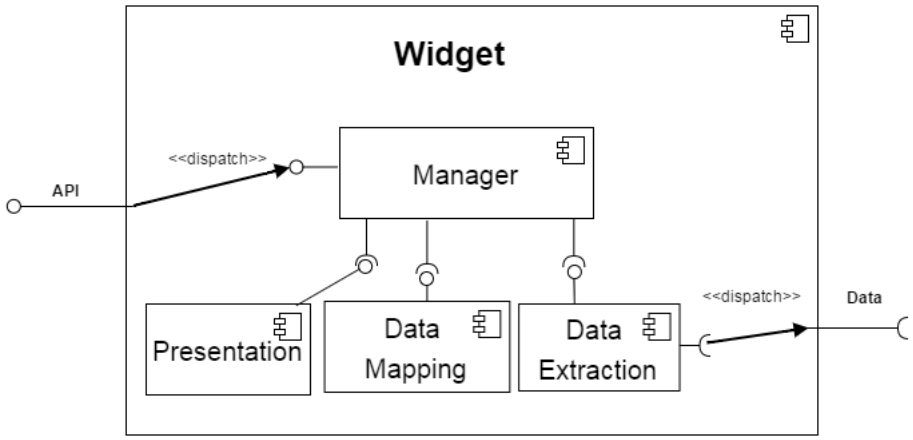


Figure 6.5: Widget internal structural relationship

Code generation for the Data Extraction The data extraction returns the information on the Target-Platform database, to be presented by the widget. It is based on the notions regarding the complementarity nature between one fact table and the dimensions, as explained on the Domain Analysis (subsection 2.1.5.5). The extracted data conforms with the model instantiated by the end user, where:

- Each Descriptive Block of the model, mapped on the properties of widget, represents the data-set labels;
- Each Measure Block of the model represents the numeric values of the data-set;
- The data set is filtered based on the conditional expressions of the Condition Blocks;
- The data set and the widget presentation is customized based on each Function Block.

In order to the widget present the data set, conformed to the model instantiated by the end user, an important step is to generate the SQL query. The query generation is delineated in the following steps:

1. **SELECT clause generation:** For this clause the generation process inspects the model instantiated by the end user and projects the Data Blocks added on the Specific Properties. In addition, for each measure, it declares the aggregation function;

2. **FROM clause generation:** In this phase, the generation process declares the relational entities that participates in the query. It iterates the selected Data Blocks and identifies its source tables;

3. **WHERE clause generation:** In this clause, the process inspects the meta-data file to find the entities from each Data Block mapped into the Specific Properties. Then, it finds the keys that defines the relation between each entity. For example, the foreign-key and primary-key that relates the measure's fact table with the label's dimension table.

4. **GROUP BY clause generation:** This clause will define the data-set granularity. Typically, each label declared on the SELECT clause is also stamped on this clause;

5. **HAVING clause generation:** In the presence of condition blocks, the query is refined, extending it with the HAVING clause declaring the conditional expression;

6. **ORDER BY and LIMIT clause generation:** These two clauses are generated depending on the presence of Function Blocks, that is, the blocks from the Function container. For instance, the Top N asset depends on this two clauses to order the data in descending mode and to limit the N records of the data set.

The generation conforms with the template defined on the EGL file, as represented in the listing 6.10.

Listing 6.10: Pseudo template for query generation - EGL plus SQL syntax

```
operation generateQueryStatement(){
    SELECT [\%=buildSelectStatement()\%]
    FROM [\%=buildFromStatement()\%]
    WHERE [\%=buildWhereStatement()\%]
    GROUP BY  [\%=buildGroupByStatement()\%]
    HAVING [\%=buildHavingStatement()\%]
    ORDER BY [\%=buildOrderByStatement()\%]
    LIMIT [\%=buildLimitStatement()\%]
}
```

When the EGL file is loaded and executed by the standalone module, the output code is generated. A generic example regarding the output of the generation process is illustrated on the listing 6.11. The given example describe each clause that is generated for

the core query. It can be observed that each label, measure and its aggregation function are stamped on the SELECT clause, alongside with the identification of the relational entities, on the FROM clause, which are the sources of the selected attributes. The condition stamped on the WHERE clause is mandatory for every generated query, as it represents the natural inner join between the entities. The GROUP BY defines the granularity of the data-set, essential to aggregate the numeric values by the stamped labels. This generic example generates a set of results similar to the report example of the figure 2.8 in the domain analysis.

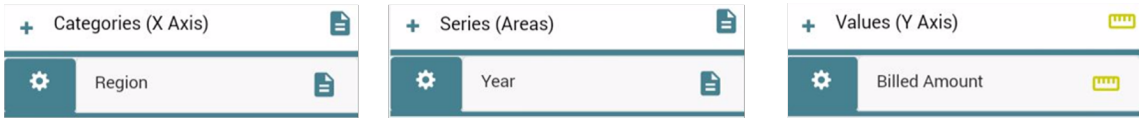


Figure 6.6: Model of a specific example: Area chart showing the billed amount by region and year. Each year is a serie, i.e, an area.

Listing 6.11: Excerpt of the query generated for the specific example of the figure 6.6, for the year 2016 serie. When executed on the Target Platform, it returns the data-set.

```
1 SELECT region as xAxisLabel,SUM(billed_amount) as numerical
2 FROM DIM_REGION as d1, DIM_PERIOD as d2, CHARGES_FACT as f1
3 WHERE f1.region_id = d1.id AND f1.period_id = d2.id AND d2.year = '2016'
4 GROUP BY d1.region
```

At this point the code to extract the data is generated. The next feature is responsible to map the data set to the visual properties of the widget.

Code generation for Data Mapping This feature is responsible for mapping the extracted information to the visual properties of the widget, as illustrated in the figure 6.7. Since, the Target-Platform depends in some third-party tools to display the widgets - namely Highcharts - this feature needs to access its APIs, as illustrated by the last method on the listing 6.12.

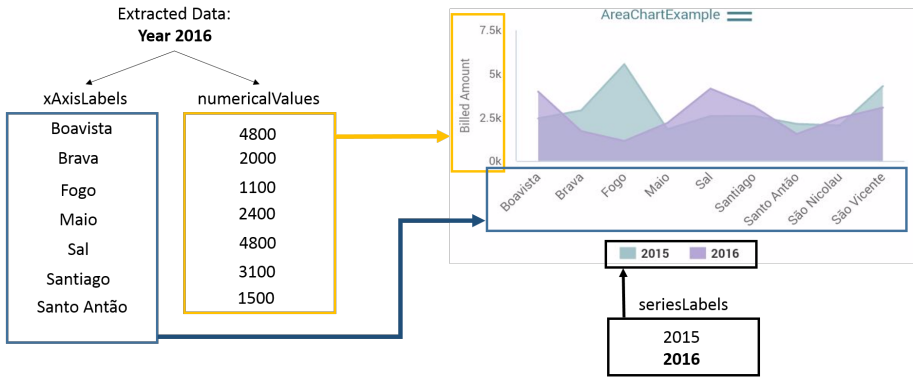


Figure 6.7: Data mapping process for the area chart example, when executed in the Target Platform.

Listing 6.12: Overall pseudo template for the area chart. Including the data mapping process - EGL and Javascript syntax

```
1   /*Returns a stringified HTML view fragment which
2    * defines the widget resentation.
3    * Is called and compiled by the target platform dashboard to display the
4    * widget
5    */
6   viewFragment = function(){...}
7
8   /*When required, it triggers and orchestrates the internal process.
9   * As the final result, it draws the widget
10  */
11  process = function(){
12      extractData().then(
13              var widget = mapData(extractedData);
14              draw(widget);
15      );}
16
17  /*Asynchronous operation. Returns a promise*/
18  extractData = function(){
19      var query = [\%=generateQueryStatement()\%];
20      return database.execute(query); /*Returns a promise*/
21  }
22
23  mapData = function(dataset){
24      var areaChart = highchartsPlugIn.AreaChartConfig();
25
26      /*Mapping the extracted dataset on the area chart visual properties*/
27      foreach(serie in dataset.seriesLabel){
28          currentArea = areaChart.series.push(serie);
29          currentArea.xAxis.data = serie.data.xAxisLabel;
30          currentArea.yAxis.data = serie.data.numericValues;}
31
32      /*Presentation*/
33      areaChart.yAxis.axisName = [\%=Model.DataBlocks.findMeasure().name\%];
34      areaChart.title = [\%=Model.WidgetName\%];
35      return areaChart;
36  }
```

Asynchronism Given the data extraction and mapping features, it is important to take into account how the operations are executed until the widget is drawn, such that, they do not compromise the interaction between the end user and the Target-Platform. To avoid the situations where the Target-Platform is blocked, until the data extraction and mapping is complete, the template specifies the use of promises[10], wherein the entire widget is processed in background.

[10]https://docs.angularjs.org/api/ng/service/$q

Extending the widget features At this point of the generation process, the core features of the widget are generated. In addition, the model is inspected to verify the presence of conditions or generic assets belonging to the Functions container. In the presence of any of this elements, the widget core features are extended.

The core features of the widgets can be extended to answer specific questions or requirements of the end user. This can be achieved with the Conditions and Functions containers. As identified on the Modeling Language Design, the Condition element goal is to select the data based on the verification of conditional expressions, the Function element, on the other hand, represents generic assets which can be used to customize the widget, depending on the goals of each asset.

The code-generation process will extend the generated code for the core features. In terms of data extraction, each of the clause of the core query is extended with additional expressions, also, in some situations, new clauses or auxiliary queries can be stamped on it. In particular, in some cases the core query is extended with HAVING, ORDER BY, LIMIT clauses.

As illustrated on the listing 6.13, in the presence of condition blocks, it will extend the data extraction feature by stamping a HAVING clause, so that, it filters the core query result set. For the case of the generic assets, for instance the Top N, it extends the data extraction feature by stamping the ORDER BY and LIMIT clause, with goal of ordering the data and limiting the data set to N records.

Listing 6.13: Example of an extended SQL query - with a Condition and a Top 10 asset- which, when executed on the Target Platform, it returns the customized data-set

```
1 SELECT region as xAxisLabel,SUM(billed_amount) as numerical
2 FROM DIM_REGION as d1, CHARGES_FACT as f1
3 WHERE f1.region_id = d1.id
4 GROUP BY d1.region
5 HAVING numerical > 3400
6 ORDER BY numerical DESC
7 LIMIT 10
```

The HVSC-Y is the only asset which extends the data mapping and presentation features. It does that by duplicating the main data-visualization components, one for each year. The code generated specifies a specific data-flow. The data-flow follows these steps (also illustrated on figure 6.8):

1. Extract the last two years on database;
2. Extract the data-set for each year;
3. Map each extracted data-set to each data-visualization component;
4. Calculate the summary data-table;
5. Draw the widget.

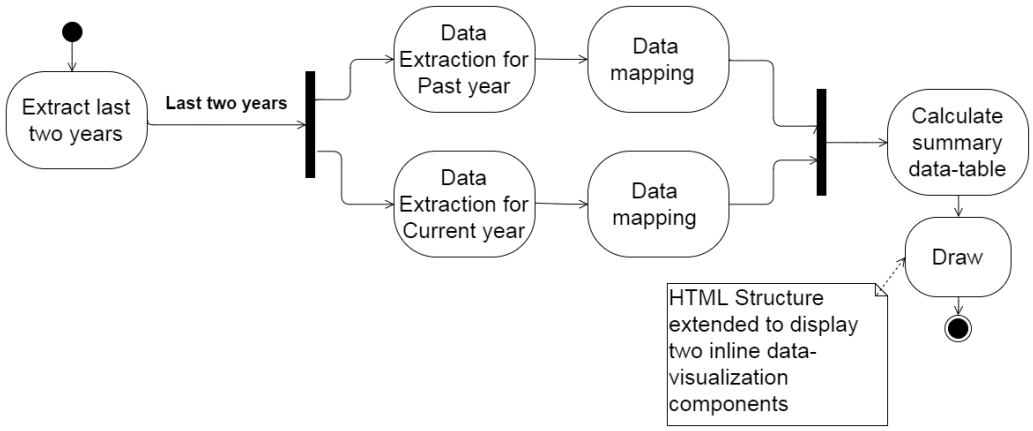


Figure 6.8: HVSC-Y asset data-flow

Design Pattern Given the characteristics of the Angular JS Framework, the widgets were structured following a Model-View-Controller (MVC) pattern. As such, the widget file has an Angular controller and service. The controller is the bridge of integration between the generated widget and the Target Platform. The controller is glued to the platform so that it receives all the inputs and events from it, namely the triggering of the internal data-flow process. The Data Extraction and Mapping features are handled by the Angular service, extracting and preparing the data for presentation. The final code generated complies with the listing 6.14.

Listing 6.14: Final widget code structure - AngularJS and EGL syntax

```
1
2  /*Angular controller logic goes here.
3  From a MVC perspective, it maintains the view updated*/
4 widget.controller("[\%=Model.WidgetName\%]Controller", function ($scope) {
5
6  viewFragment = function(){...}
7
8  /*When required by the Target Platform,
9   *it triggers and orchestrates the internal process.
10  *As the final result, the widget is drawn.
11  */
12 process = function(){...}
13
14 });
15
16 /*Angular service logic goes here.
17 From a MVC perspective, it manages the model*/
18 widget.factory('[\%=Model.WidgetName\%]DataService', function() {
19     /*Asynchronous operation. Returns a promise*/
20     extractData = function(){...}
21     mapData = function(data){...}
22 });
```

6.4.2.4 RESTFul Web-Service

Without a communication interface, the core services are not reachable by the end users. Hence, this service is responsible for mapping the core services (Model-Validation and Model-To-Code-Generation) to Uniform Resource Identifiers (URIs) that are called remotely by the client, with AJAX HTTP requests.

To realize this idea, this module was implemented based on the packages of Spring MVC Framework. In Spring's approach to build RESTFul web services, the HTTP requests are handled by controller classes. Each controller method is bound with RequestMapping annotations which, as the name suggests, are responsible for mapping web requests onto the specific controller handler methods. These handler methods dispatches the requests to the core services of the Server Component and returns the result back to the client.

As an illustration, in the listing 6.15 is a request method handler with an annotation that maps to it, the requests of type GET and with URIs such as "widgetGenerator/editor". In this example, the Editor Component view is sent to the client.

Listing 6.15: Request Editor method handler: returns the view containing the Editor

```
1 @RequestMapping(value = "widgetGenerator/editor", method = RequestMethod.GET)
2 public requestEditor(){...}
```

These handler methods, forms the API of the Server Component, and is the main support for the integration of the three main components of the architecture.

The REST API is provided in the following table 6.1, wherein, for each URI, is presented the resource which it points to.

Table 6.1: Server REST API

URI Nº	*Method:URI*	*Resource*
1	GET:URLBASE/editor	Editor page
2	GET:URLBASE/templates	List of deployed assets
3	GET:URLBASE/<templateURL>	Selected template meta-model
4	POST:URLBASE/generate/<templateURL>	Call for model-validation and code-generation services. The model instance must be submitted. It returns the process final state
5	GET:URLBASE/download/<widgetName>	Returns the generated widget

URLBASE = <machine>:<port>/widgetGenerator

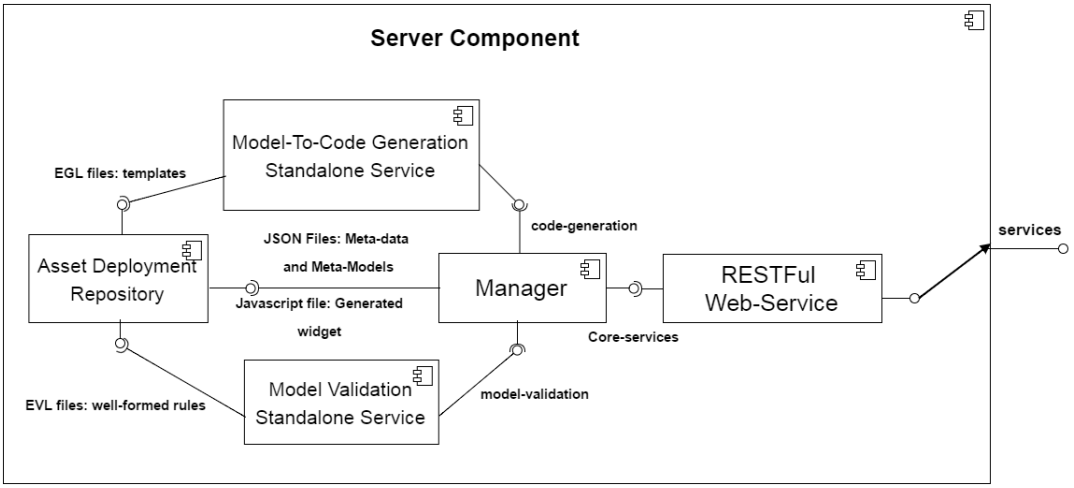


Figure 6.9: Overview: Server internal structural relationship

6.4.3 Target Platform extension

This is the target platform for the generated widgets and where the user monitors the corporate data.

As mentioned in the section 1.1 of the introduction chapter, the company had already developed this mobile application for a Mobile Business Intelligence project, in which is presented a set of pre-defined dashboards composed by analytical components. However, because the analytical components are pre-defined and static, it does not provide features to configure new widgets nor to compose the dashboard based on the user preferences and demands.

Hence, the Target Platform was extended to provide to the end user the features that supports him to perform his role, that is, to monitor the corporate data by means of visualization and interaction with analytical widgets at same time with the freedom to compose its own dashboards. To provide this capability the Target Platform needed to:

1. Provide to the end user the capability of interaction with the dashboard, based on adding, dragging, resizing and removing actions over the widgets;

2. Persist the changes made on the dashboard;

3. Provide to the end user the ability to configure new widgets by requesting and displaying the Editor view to the end user.

Inspired on this ideas, we started by developing a local Widget Repository on the Target Platform to store and present the imported widgets. It is essentially, a list of widgets where the end user can preview and add them on the dashboard or remove totally from the device.

When added to the dashboard, the end user is capable of drag, re-size or remove the

widget instance. These actions are possible thanks to the grid system behind the dashboard. The grid system is based on the Gridstack library [11], it implements a drag-and-drop multi-column grid on the dashboard view, enabling the end user to interact with the widgets by dragging and resizing actions, and automatically managing the collisions between this components by re-configuring their arrangement. The configuration of each widget instance, in terms of position and dimension, is persisted on database.

In order to display the widget, either on the Dashboard or in the Widget Repository view, the Target-Platform access the respective generated API to trigger its data extraction, mapping and drawing processes.

6.4.3.1 Persistence

The stored information is denominated as Widget Meta-Data and describes the imported widgets and their relationship with dashboards. As represented in the Entity-Relations diagram in the figure 6.10, the entities that manages this meta-information are:

- **Dashboard:** It stores meta-data related to the dashboard, like its name;
- **Widgets:** It stores meta-data related to all widgets installed on the device, namely:
  1. **ID:** Unique identification of the Widget, generated sequentially on widget importing and installation;
  2. **Name:** Name of the widget;
  3. **Controller:** Name of the AngularJS controller function responsible to draw the widget. The controller function resides in the generated Javascript file and is lazily loaded when the widget is selected;
  4. **Path:** Path where the generated Javascript file resides. Necessary for the component which loads the file in lazy mode;
- **Dashboard Widgets:** It represents a Many-to-Many relation between the Widget and the Dashboard entity and stores the meta-data of Widgets allocated on Dashboards, namely:
  1. **WidgetID:** ID of the widget;
  2. **DashboardID:** ID of the dashboard where the widget resides;
  3. **X Position:** Horizontal coordinate of the widget, on the grid structure of the dashboard;
  4. **Y Position:** Vertical coordinate of the widget, on the grid structure of the dashboard;
  5. **Width:** Width dimension of the widget;
  6. **Height:** Height dimension of the widget;

[11] http://troolee.github.io/gridstack.js/

7. **RowID:** Sequential ID;

Based on the information stored on the Widgets relational entity, the Target-Platform is able to load the widget saved on the mobile device storage and present it to the user. Nonetheless, because the application is web-based, it does not have the direct access to the mobile device API. So, to install, remove or load the widget file, the application uses the Cordova native bridge features and permissions.

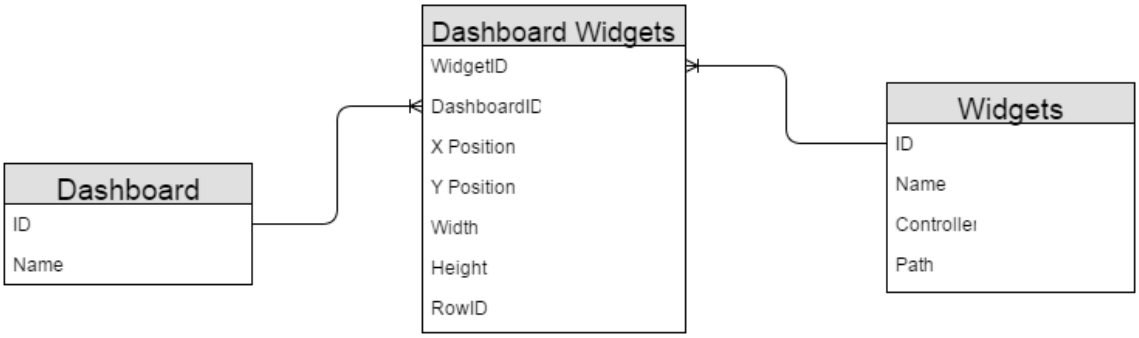


Figure 6.10: Widgets Metadata

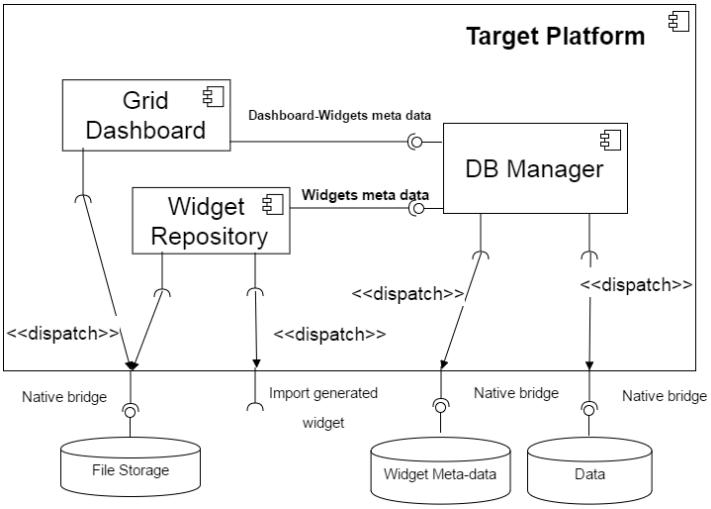


Figure 6.11: Overview: Extended Target Platform structural relationship

6.5 Examples

In this section a set of examples is given, in order to review the main components description given on 6.4.3, 6.4.2 and 6.4.1 and to give an overview regarding their integration and interaction.

6.5.1 Requesting the Editor Component view

Based on the illustration depicted on the figure 6.12, the Editor view is presented to the end user by making a request with the URI Nº1. The Server Component returns the

Editor view and is presented to the user in the Target Platform within an Iframe element.

6.5.2 Generating a widget

Based on the illustration depicted in the figure 6.13, the widget is generated on submit event, triggered by the end user.

Based on the widget type, previously chosen by the end user, a widget generation request is sent to the Server Component with the URI Nº 4 and is dispatched to the EVL and EGL standalone modules, responsible to load the well-formed rules and the templates files and, respectively, to execute the model-validation and code-generation processes.

The final result is either a set of validation messages or a success state, depending on the results of the model-validation and the final state of the code-generation. Then, the generated widget file can be requested and saved on the mobile device.

6.5.3 Saving the generated widget

This example is illustrated in the figure 6.14. From an end user point-of-view it saves the generated widget, previously modeled in the editor. Internally, it downloads the generated widget based on requests upon the Web-Services and saves it in the local Widget Repository in the mobile device.

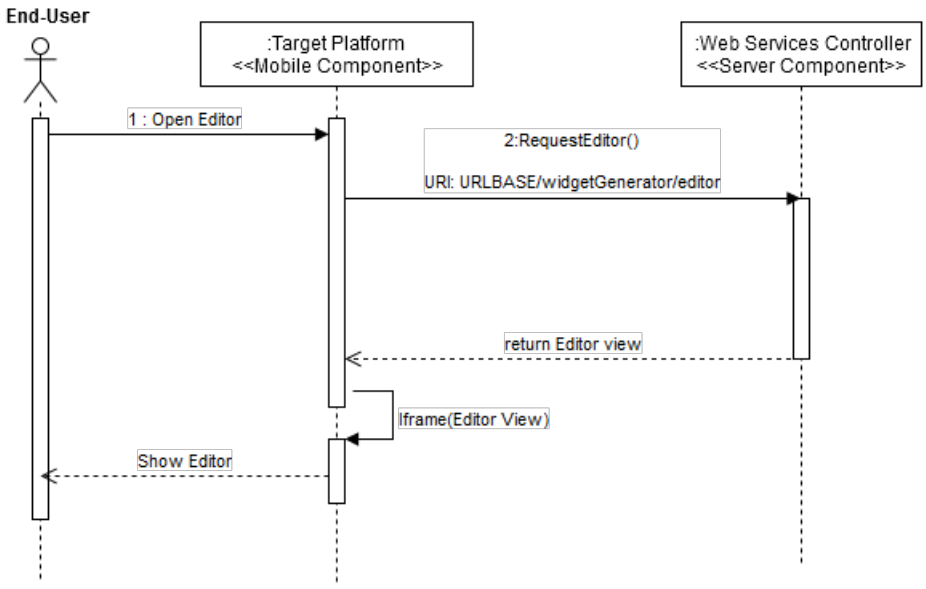


Figure 6.12: Requesting the Editor Component view

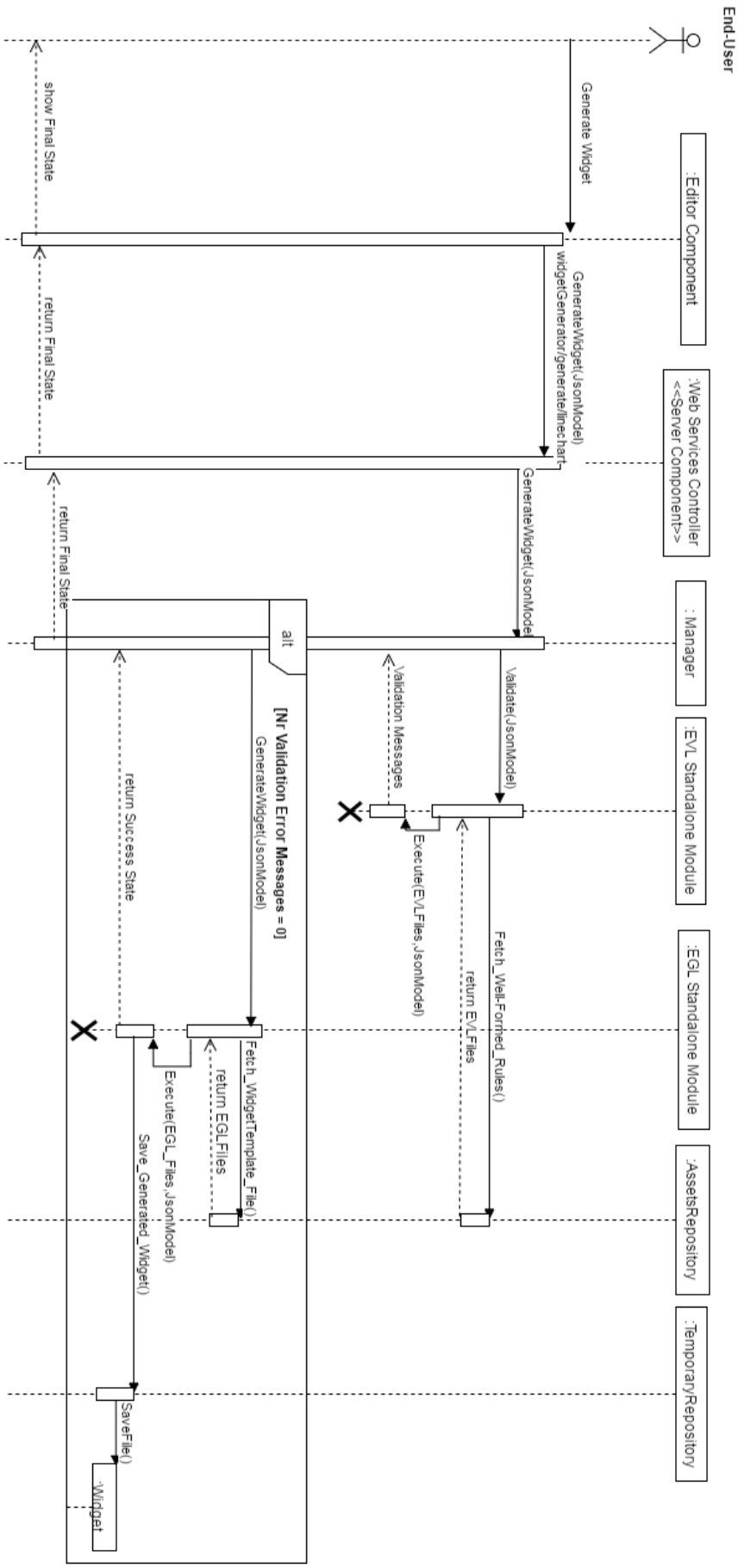


Figure 6.13: Generating a Widget

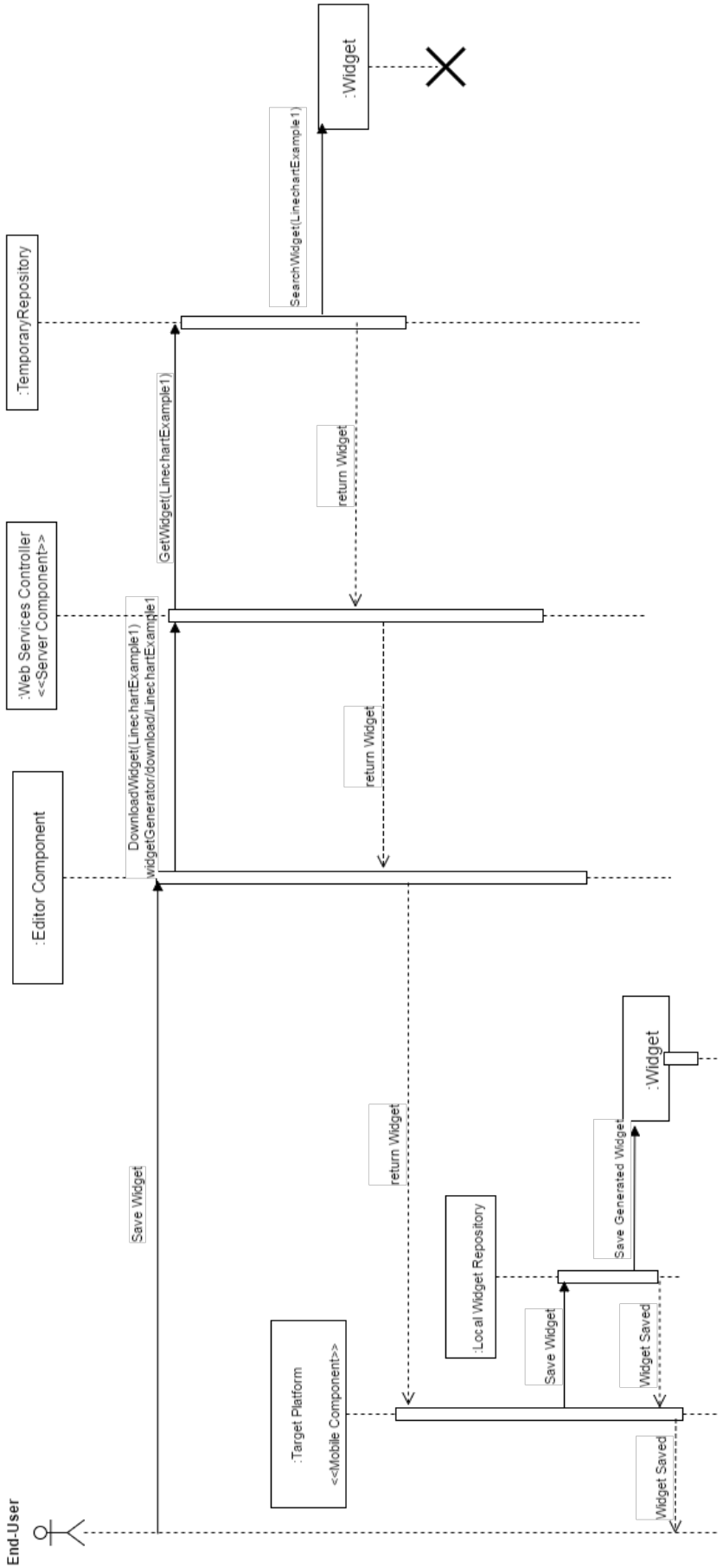


Figure 6.14: Saving the Widget

6.6 Summary and discussion

As final thoughts regarding the implementation, we can conclude that the conceived architecture aligns with our initial idea - to design a mean of cooperation between the main actors, inspired by the Software Product Line concept described on 2.1.4. The Product Line metaphor can be applied to conceptually describe the architecture and the cooperation between the actors.

The product line scope is maintained by the domain experts, regarding the business requirements and needs of the end users. The generic production assets are deployed on the Server and supplied for product development. The product development is performed by the end users on the Editor Component, with the goal of generate the final product, that is, the widget. This cooperative strategy brings advantages when it comes to situations where the client wants to generate a certain type of widget with less configurations as possible, because it will be defined, by the product line developer a specific asset which is aligned to that certain question. This was demonstrated with the example of the HVSC-Y asset.

To implement this idea, we have inspired on the Model-Driven concepts to rise the level of abstraction, such that, the end users could configure and generate the widgets, based on a Editor Component and a Modeling Language which is specific to the Domain of BI. The Server Component is the kernel, being responsible to handle this transformation process and, based on web-services manage and display the production assets deployed by the domain experts. As seen, the assets are: The meta-data; The widget meta-models; The templates implemented on EGL syntax, alongside with the well-formed rules, which are implemented on EVL syntax.

These assets were designed to be horizontal or vertical to the business concepts and requirements. For instance, every widget type is generic, in the sense that, for each one, it is provided templates and meta-models describing the generic properties, which are customizable based on the mapping of the meta-data attributes. The meta-data is vertical to business concepts, because it describes the corporate information that resides on the database. So, in business related changes, it is most likely that the meta-data needs reconfiguration efforts from the domain experts.

In addition, the generic assets represented by the Function container, can also be seen as vertical or horizontal production elements. For instance, the HVSC-Y asset was designed, keeping on mind the typical analysis which are time-based. This asset, in the best case, can be, also, allocated on other business contexts. On the other hand, in the worst case, this asset is not valid, namely on contexts where the data-model does not includes time dimensions. Whereas, for instance, the TOP N asset is more generic and can be used in more analysis, regardless of the dimensions.

Nonetheless, it is still important to evaluate if the prototype is capable of answering the research question. Therefore, without the validation, the prototype development

would still be incomplete. The next chapter will cover this aspect by explaining the validation process, basing on empirical techniques from Human-Machine Interaction to evaluate the modeling language.

7

Validation

Usually the validation of Domain-Specific Languages via experimental usability is neglected by Software Language Engineers [Ped10], considering only the design, implementation and deployment stages. The authors in [A B12b] and [A B12a] proposes that, the empirical techniques used on human-machine interaction can also be used to evaluate the Domain-Specific Languages.

To analyse and evaluate if the prototype is capable of facilitating and empower the user on its organization monitoring activities and to identify future improvements, an usability study was fundamental. Usability is defined by ISO 9241-11 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". This definition consists of three main parts: the effectiveness, efficiency and satisfaction. The effectiveness is measured by the amount of errors made by the subjects, the efficiency is measured by the task completion time and, on the other hand, the satisfaction level can be extracted by subjects answering to open-ended questions or rating scales [FHH00]. In particular, this tests rests on the extraction of time, error-rates, task-completion-rates, open-ended questions and rating scales.

To perform this tests we needed to define a series of tasks and collect a set of end users and domain experts to represent the subjects. In particular, it was interesting to perform the usability tests with business users and experts on Business Intelligence (BI) from the company. Since that, it is important to take on account the subjects availability, the biggest constraint appeared to be the gathering of real-life business users, that is, the clients from the company. Hence, we prepared a usability test based on a simple and generic case study that would fit any type of user, enabling him/her easily comprehend and solve the business related questions.

7.1 Prepared material for the experiments

Before the experiments, it was prepared a set of materials to be used by the recruited subjects. Namely, contextual information about the usability test, divided in a brief summary of the Editor, its goals and features, alongside with an explanation of the case study example. This information could be consulted during the experiments execution. Additionally, it was available a tablet device, to run the Target Platform and the Editor. Its specifications can be found on table 7.1.

Table 7.1: Tablet device hardware and software characteristics, used in the experiments

Hardware	*Details*
Platform	Android
Platform version	5.0.2 Lollipop
Resolution	800x1280 px
CPU	Quad-Core
Storage	16 GB
RAM	2 GB

7.2 Evaluation Procedure

The usability test was conducted based on the following phases: Users interview; Training session; Case study presentation; Execution of empirical tests on the end users and domain experts of the organization; Satisfaction questionnaires;

7.2.1 Users Interview

The participants were interviewed in order to gather their relevant information. This profile data, would enable to us to distinguish the two types of subjects: The end user and the BI domain expert. The results of the interview are summarized on table 7.2 (section 7.3).

7.2.2 Training Session

After collecting the user profile data, the Target Platform and the Editor were presented, and a quick training session was given. To ensure that the participants would not be exhausted during the session, it was conducted in an interactive way, in the sense that, the user could, at any time during the training, place questions and give his opinions. The training started by presenting a set of pre-defined widgets on the Widgets Repository and running an example where the evaluator configured a simple dashboard by adding, dragging, resizing and removing the widgets. At this stage, it was clearly that, for some

participants, despite the configuration empowerment given to the end user based on the interactivity between the widgets and the dashboard, it was necessary to configure and generate new widgets. For this cases, and to answer the question, the editor was introduced alongside with the explanation of the Modeling Language, the relevant features and its functional goals.

7.2.3 Case Study Presentation

To finish the training session and to consolidate the knowledge given to the participants, they started by reading the case study, and a brief summary of the Editor. The case study used is the same as the one presented on chapter **??**.

7.2.4 Empirical Tests

In the next phase, the participants answered a total of four multiple choice questions. These questions would enable us to understand if the participants had understood the concepts, the goals and specific features from the Editor and its Modeling Language. In particularly, we were interested in:

- Verifying if the user understood that, each widget template has its specific properties;
- Verifying if the subjects could relate the domain of the problem with the solution by matching the presented model with the correct solution;
- Understanding if the user comprehended the goal of the Functions container;
- Understanding if the user understood the Top N filtering function;
- Verifying if the user understood the goal of the Conditions container.

Finished the multiple choice questions, the subjects answered two practical exercises. The results of the practical exercises would enable us to analyse if the participants could reach the expected solutions and if they understood the specific features and concepts from the Target Platform, the Editor and its Modeling Language, measure the time and error rates during the tests. In particularly, we were interested in:

- Verifying if the subject could navigate from the Target-Platform to the Editor and vice-versa;
- Verifying if the subject could solve the problems using the Modeling Language;
- Identifying the number of mistakes that the subjects did during the experiments;
- Verifying if the subject could recognize and recover from possible existential errors;
- Measure the time to accomplish the tasks.

During the practical experiments, a software tool recorded the device screen to identify and analyze the participant's errors and solutions and to measure the time taken to solve the problems. The tasks provided to the users are detailed on 7.4.

7.2.5 Satisfaction Questionnaires

In the next phase, a closing questionnaire was presented and answered by the subjects. This questionnaire was structured based on the user profile, gathered on the Users Interview phase. In this sense, it was constructed two closing questionnaires:

- **One questionnaire targeted to the end users:** The goal of this poll is to extract the satisfaction level of the participant, after using the prototype, ask and identify the problems and obstacles that she/he encountered during the experiments;
- **One questionnaire targeted to the domain experts:** The goal is to extract insights and recommendations from the expert regarding the prototype.

Thus, although the domain experts executed the tasks, it was only to contextualize them with the prototype. The goal was not to evaluate the usability, since they are not the target users of the modeling language. Instead, the goal is to extract their insights and recommendations so that we can validate the prototype regarding the principles of BI. The figure 7.1 summarizes the evaluation procedure.

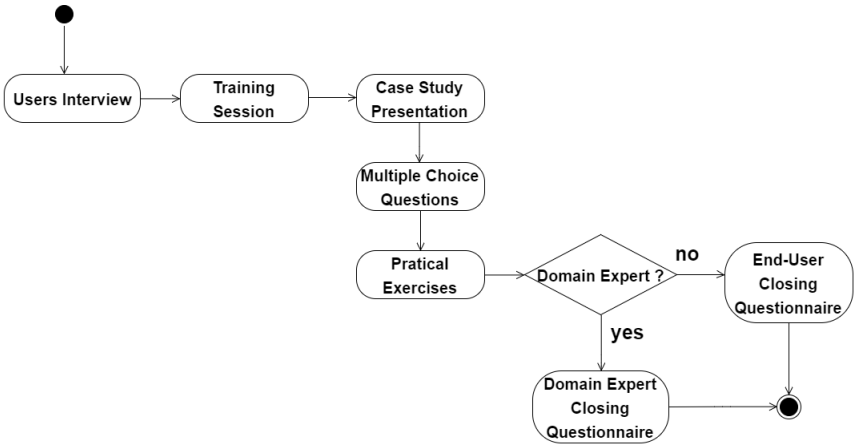


Figure 7.1: General procedure for the modeling language evaluation

7.3 The Subjects

The subjects were collaborators from the company with a diverse background of knowledge. They participated, voluntarily, summing a total of 14 individuals.

Five of the participants are from computer science, where two are currently working on the BI domain. Three subjects are from electrical engineering area where none is specialized on BI. Three participants were from industrial management area and, similarly,

none of them is specialized, or is working, in the BI area. The others three subjects are, equally, distributed on the architecture, mechanical engineering and biomedical areas, where the subjects from architecture and mechanical engineering are currently specialized and working on the BI domain. The participants where equally distributed by their graduation degree - 50% of the total participants had a Bachelor and 50% had a Master degree.

Table 7.2: The subjects

Course	*Number of Subjects*	*Aprox. Percentage*	*Bachelor (B) / Masters (M)*	*Subjects Working or Specialized on BI Domain*
Computer Science	5	36%	3 B; 1 M	2
Electrical Engineering	3	21%	3 M	0
Industrial Management	3	21%	2 B; 1 M	0
Mechanical Engineering	1	7%	1 M	1
Architecture	1	7%	1 M	1
Biomedical	1	7%	1 B	0
Total	14	100%	7 B; 7 M	4

Summing up, the domain experts group counted, in total, with four subjects. In the other hand, the end users group counted with the total of 10 participants.

7.4 The Tasks

7.4.1 Multiple Choice Questions

7.4.1.1 Question 1

Consider the figure 7.2. To the containers Categories (X- Axis) and Values (Y- Axis) were added attributes. On the X Axis container, the month was added and on the Y Axis the Billing. The remaining containers are empty.

Which widget will be generated?

1. **Monthly evolution, of the billed amount. Depicted by a line chart**;
2. Monthly evolution, of the billed amount. Depicted by a line chart, where the lines of the chart represents each month;
3. Monthly evolution, of the billed amount. Depicted by a bar chart;
4. Billed amount by region (location). Represented by a line chart.

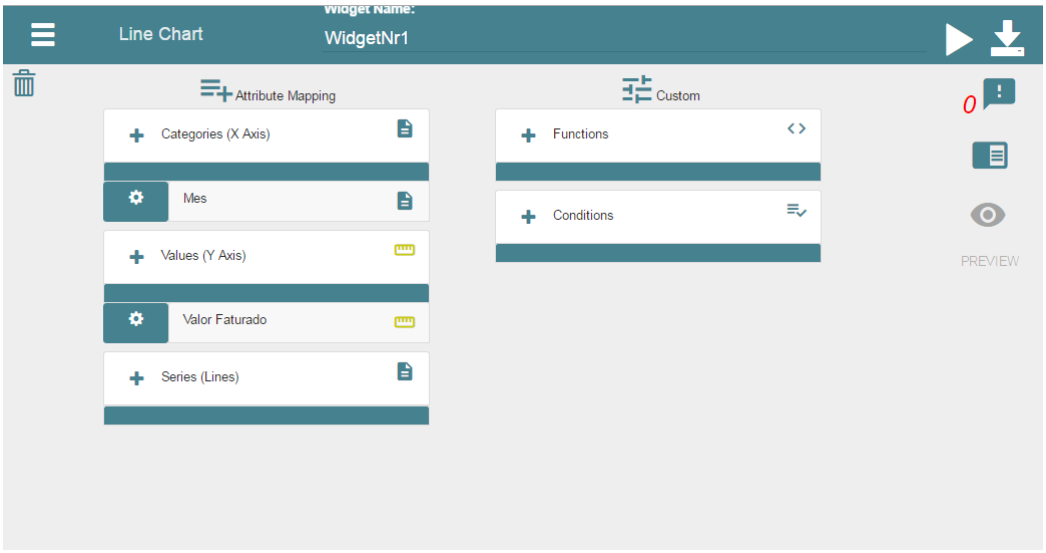


Figure 7.2: Multiple choice: first question

7.4.1.2 Question 2

Now consider that the user chose to build a bar graph which shows the amount billed. Additionally, he has chosen a function, as shown in the figure 7.3 Which widget will be generated?

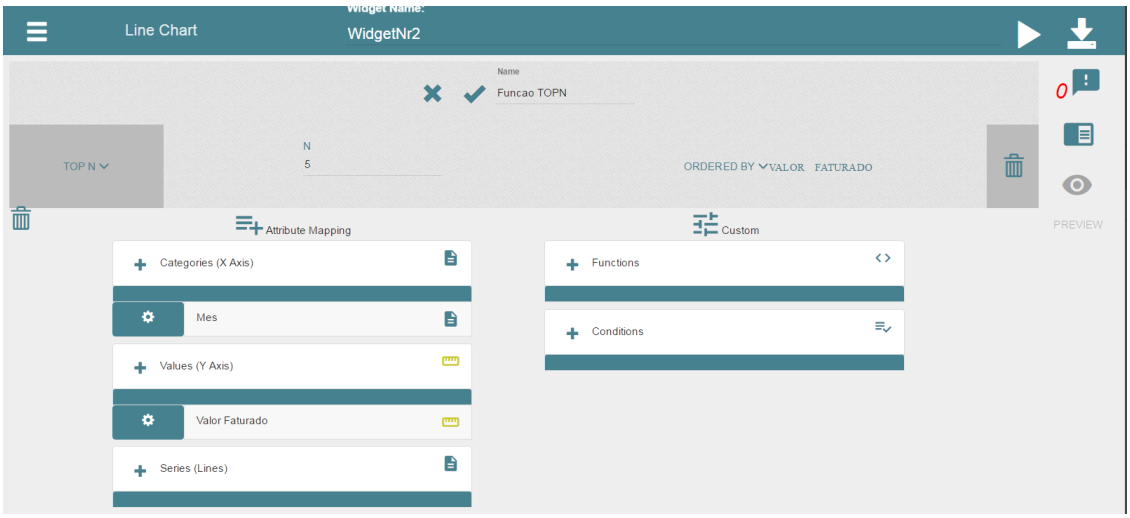


Figure 7.3: Multiple choice: second question

1. A bar chart with the billings temporal evolution, by month;

2. A bar chart with the billings temporal evolution, by month and only for May;

3. **A bar chart with the billings temporal evolution, by month, presenting a classification with the five months with highest billed amount**.

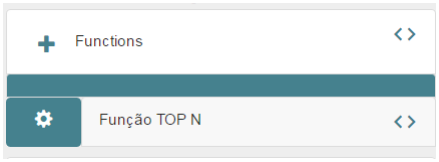


Figure 7.4: Functions container after confirming the selected function

7.4.1.3 Question 3

In this question, please consider that you want to set up a heatmap to show the distribution of the amount billed by Sector and only for the month of August.

A heatmap is a matrix in which the cells have metrics, the rows and columns have the categories.

In this sense, it is pretended to have the month of August as the row and Sector as the column of the matrix.

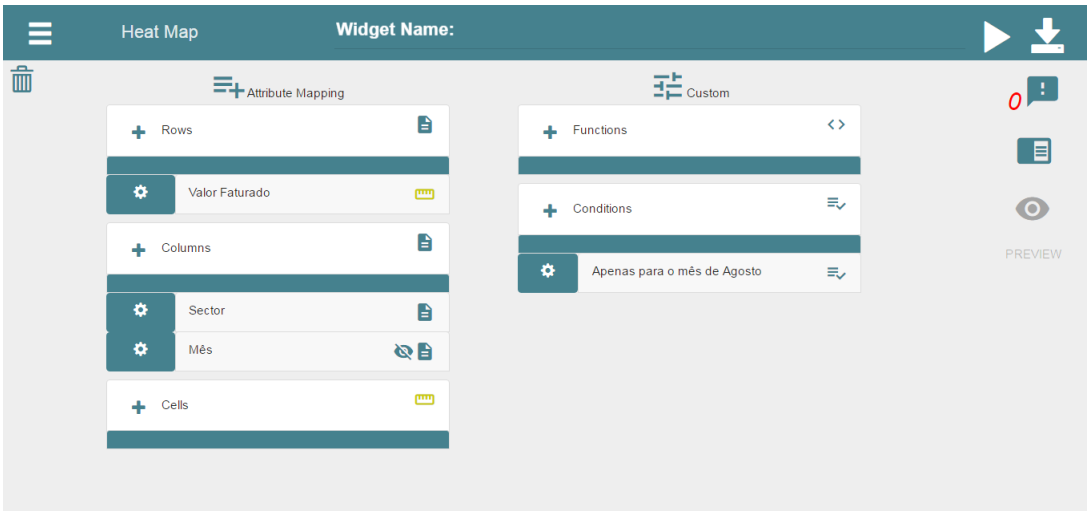


Figure 7.5: Multiple choice: third question

Regarding the aforementioned goal, is the model presented on the figure 7.6 valid? If you asked negatively, please point the errors.

1. Yes;
2. **No.**

7.4.1.4 Question 4

Imagine that you are asked to build a dashboard with graphs represented in Figure 5. Each option you would choose to compose the dashboard?

1. Build three widgets of the type donut-chart;
2. **Build a widget of the type combined with three donut-charts.**

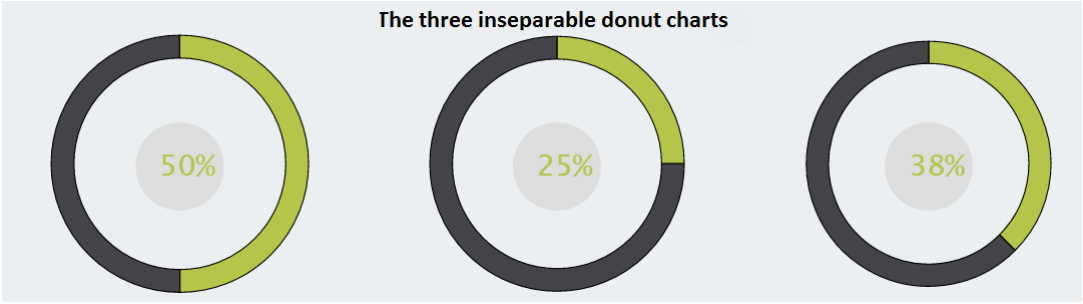


Figure 7.6: Multiple choice: fourth question

7.4.2 Practical Experiments

7.4.2.1 Question 5

In this question, let us suppose that you belong to a specific department of a fictitious Utility company. In particular, the Utility name is *Eletrão* and you belong to the department against the fraud and debt. As a rigorous and concerned business person, you want to monitor which three regions has the biggest debt values. Given this scenario, please create a model of a bar chart and other widgets that you find relevant to answer the question. As a final step, configure the dashboard.

7.4.2.2 Solution

On this question is expected, from the subjects: To map the Region and Debt variables to the X Axis and Y Axis, respectively; To choose and configure the Top N asset from the Functions container. The configuration of the Top N asset is expected to be: $\boldsymbol{N} = 3$ and ***Ordered By*** = *Debt*.

7.4.2.3 Question 6

In this question, let us suppose that you are the CEO of *Eletrão*. On the beginning of the year 2016, you had defined and invested on a plan to minimize the debt from the clients – that is, the clients that do not pay their bills – and decided, as a strategical plan for the company, to raise the profit of the amount charged to clients to, at least, 50 % in relation with the past year.

Supposing that, it remains only one hour to the end of the year 2016, as a rigorous and concerned business person you will want to know, with a widget, if you reached the goal.

Given this scenario, please model the line chart and other widgets that you find relevant to answer the question. Based on the widget(s) you have created and as long as you are the CEO of the company, do you have reasons to be concerned?

7.4.2.4 Solution

On this question is expected, from the subjects: To map the Month and the Profit on the X Axis and Y Axis, respectively; To extract the Profit by specifying the difference between the Billing and the Debt; To choose the Homologous VS Current By Year asset (HVSC-Y).

7.5 End user test results analysis

The following results were extracted from the tests made to the end user group, which counted, in total, with 10 subjects.

7.5.1 Multiple choice questions results

The multiple choice questions enabled us to verify if the subjects understood the Modelling Language and could map its concepts on real-world problems. The results extracted from this tests suggests that the subjects, apparently, understood both the questions presented and the concepts from the language, as no wrong questions were found.

However, is important to note that, such results do not prove that they really understood every aspects. For instance, regarding the Question 3 - 7.4.1.3 - only one participant did answer affirmatively when asked if he/she know what a heatmap is. After the explanation, everyone responded to the question correctly: The model presented was not correct. However, some of the participants responded wrongly when required to signal the incorrect configurations.

Table 7.3: Wrong responses

Answer Nº	Wrong Response
1	Billed Amount should be Year.
2	The Month should be on the Row and the Billed Amount on the Columns.
3	Billed Amount should be on the Cells and, either the Sector or the Month should be on the Rows.

The answers submitted by the subjects suggests, in fact, that, either, they did not understood the question or could not associate the heatmap characteristics with the presented model. For instance and considering the samples of wrong answers of the Question 3 - 7.4.1.3 (chapter 7) -, presented on the table 7.3, the first one is an indicator that the participant did not understood the presented problem, as the question did not mentioned the year as an attribute to include on the widget. The second response suggests that, although the subject understand that the month should be on the Rows, he/she did not really understand that the Billed Amount should not be located on the column, but

on the Cell of the heatmap. The final sample, suggest that the subject did understand and mapped the characteristic of the heatmap to the presented model. Nonetheless, he/she did not really understood or captured the essential part of the question, as it was explicitly mentioned to consider the Month as the row of the heatmap.

7.5.2 Practical experiments results

The practical experiments enabled us to evaluate the Modelling Language in terms of usability. As a result of the proceeded tests, we extracted the time measurements and task-completion rates to analyze, respectively, the effectiveness and efficiency. Lastly, the satisfaction was extracted.

7.5.2.1 Analysis and findings concerning the Efficiency

As represented on the chart of figure 7.7 and 7.8(a), it can be observed that, on average, the total time spent on the Question 5 - 7.4.2.1 (chapter 7) - was bigger that the time spent on the Question 6 - 7.4.2.3 (chapter 7). Considering only the success cases, this evidence stays unchanged, as represented on the chart of figure 7.8(b). The reason for this was mainly due to the configuration mistakes and the difficulties expressed to reach the solution for the Question 5.

A particular case was what happened to the Subject C. The large amount of time spent on the Question 5 was due to the mistakes made when modelling the widget and slips made when interacting with the Editor interface elements. The subject affirmed that himself had fumbled on specific features of the Editor, which led him to the wrong solution, and also pointing out some user interface problems, namely some aspects that were not well suited on mobile contexts, which will be further analysed on the effectiveness analysis.

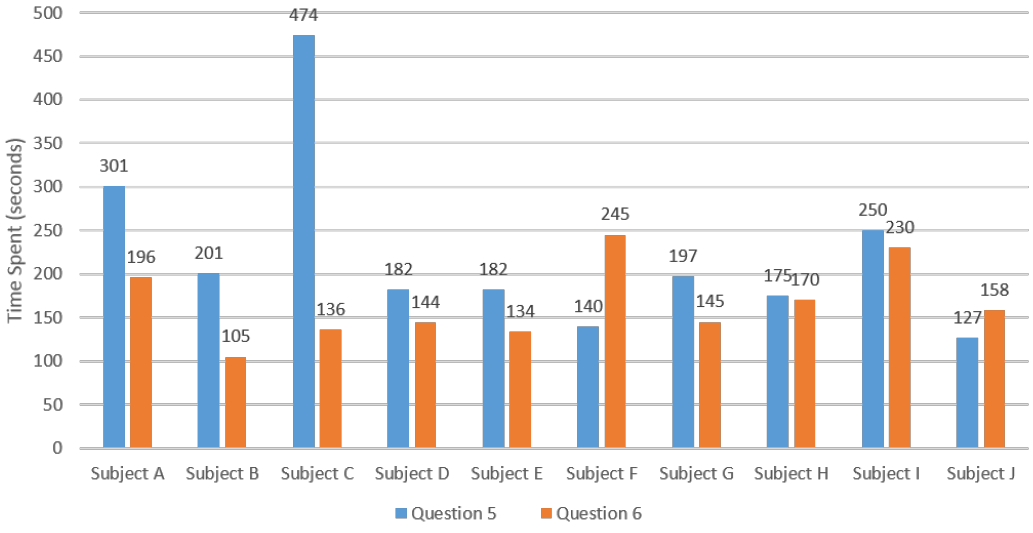
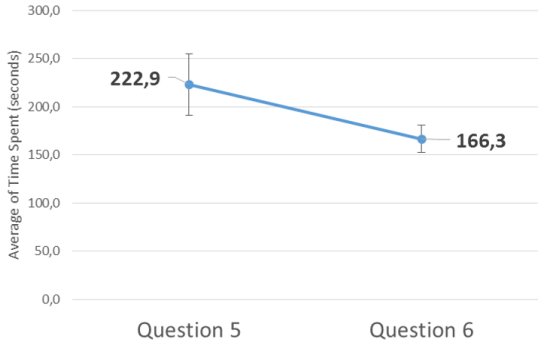
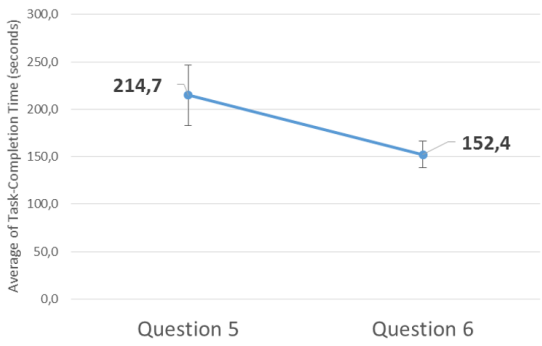


Figure 7.7: Time spent for each subject



(a) Average time spent per Question



(b) Average task-completion time per question (Unsuccessful scenarios excluded)

Figure 7.8: Time measures, with error bars

7.5.2.2 Analysis and findings concerning the Effectiveness

To better understand the mistakes and slips made the subjects, an analysis was configured, regarding the task-completion and the quantity of errors.

Task-Completion

In order to measure the task-completion rate, a list of task-completion categories was defined:

- **Easy:** Success on first try, without any difficulty;
- **Medium:** Success on the first or second try, with difficulty expressed by the subject;
- **Hard:** Success on the second or more tries, with difficulty expressed by the subject;
- **Assisted:** Solution directly given by the evaluator;
- **Fail:** Subject failed or gave up.

The success was grouped into three classes. In the category **Easy** resides the cases where the subject answered the question in the first try without any difficulty, recognizing, immediately, the right Modelling Language elements to configure the solution. On the other hand, based on the number of tries and the difficulty expressed by the subject to provide a solution, the success is grouped into two different types: the **Medium** and the **Hard**. Note that, the number of attempts is counted given the number of times the end user triggers the code generation process until he/she reaches the correct solution. Typically, the ideal scenario would be the case where the subject reaches the solution correctly with no validation messages. An example of unsuccess is when the server returns an error message from the model validation process.

In the class **Assisted** resides the cases where the researcher/evaluator, helps the subject by directing him to the solution. This scenario happens when the subject seeks for

support from the evaluator, given the observed difficulties, emotional state or inexperience. Although the task could be completed, it is not considered to be successfully completed.

In the class **Fail** resides the cases where the subject generated a widget but failed to respond the question or simply gave up, not completing the task successfully, due to the difficulties, emotional factors or inexperience.

Keeping this analysis configuration on mind, the success and fail rates, for each question, were calculated based on the percentage conversion of the ratio between the number of occurrences of a specific task completion class and the number of task undertaken, therefore:

$$SuccessRate = \frac{Easy \quad + \quad Medium \quad + \quad Hard \quad Occurrences}{Total \quad number \quad of \quad tasks \quad undertaken} \times 100\%$$

and,

$$FailRate = \frac{Assisted \quad + \quad Fail \quad Occurrences}{Total \quad number \quad of \quad tasks \quad undertaken} \times 100\%$$

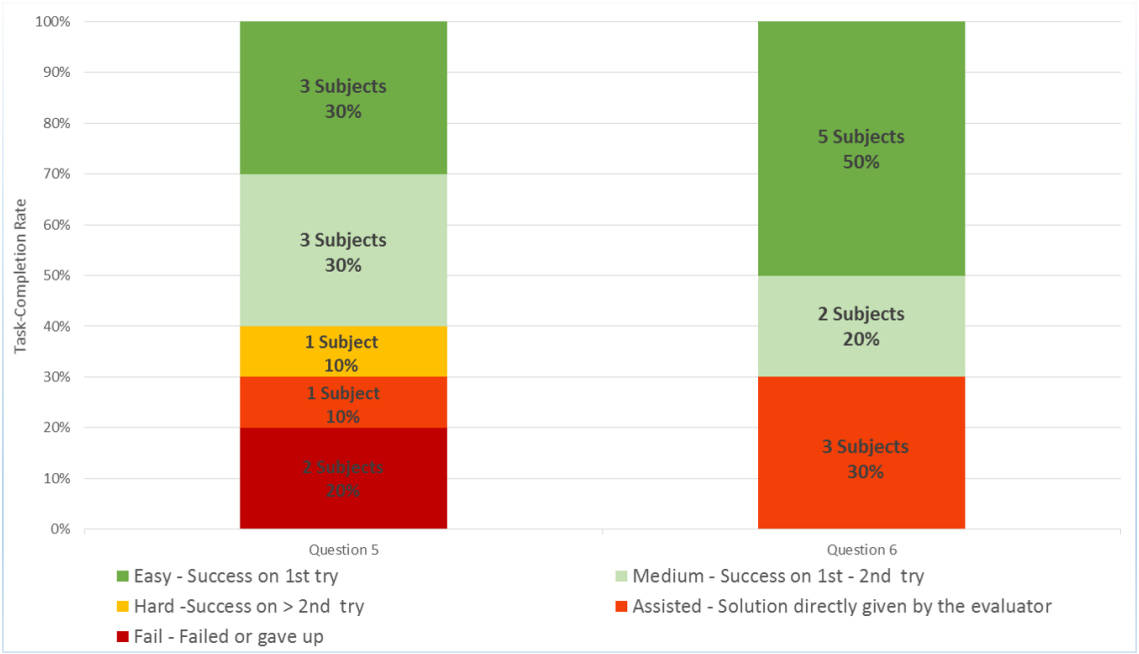


Figure 7.9: Task-Completion Rates

Based on the results displayed on the chart of the figure 7.9, and considering that the Assisted cases are unsuccessful scenarios, it can be verified, from the Question 5 to the Question 6, that the effectiveness does not change. The Question 5 and the Question 6 counts with 70% of successfully completed tasks, and by the other hand, 30% of unsuccessfully or not completed tasks. However, when analysed individually, each task completion class, it can be observed some differences on each task-completion class.

On the Question 5 can be verified that 20% of the executed tasks, that is, two subjects from the 10 end users, have failed to complete the task or gave up. In this cases was observed that both the subjects could not identify the problem variables neither map it to the Modelling Language. Additionally, it was observed that some subjects, besides identifying the Top N as the solution for the problem, they actually made mistakes when configuring the asset - this mistakes are further explained based on the errors analysis.

From the Question 5 to the Question 6, it can be verified an increase of subjects who completed the task Easily, in contrast with the reduction of Failed, Hard and Medium cases. The justification is that, in opposition with the Question 5, it was observed that the most of the subjects identified and mapped correctly the variables of the problem, had learned from the previous mistakes and identified, almost or immediately, the Homologous VS Current Year (HVSC-Y) asset as the solution for the problem. The Failed cases are null, in contrast with the rise of Assisted cases. This evidence aligns with the observed situations where the subject did not recognized the Functions container and the goal of the HVSC-Y asset, requiring the assistance from the researcher to achieve the correct solution.

Errors

Given the task-completion rates analysis, it was also important to identify the mistakes made by the subjects to recognize the user interface problems and unexpected actions that potentially limited the user to reach the correct solution with minimum difficulties and consumed time resources.

Therefore, the mistakes made by subjects were generalized in the following categories:

- **ERR1:** Incorrect interactions with the Editor interface. For instance, touching on non interactive sections;
- **ERR2:** Wrong widget configurations, for instance choosing Month instead of Regions for the X-Axis of the Line Chart or choosing a wrong parameter value;

And aggregated across the following severity levels:

- **Severity 1 (SEV1):** An error which, by itself, did not drove to a wrong solution, nonetheless, contributed to an increase of time resources;
- **Severity 2 (SEV2):** The error led to a wrong solution or unexpected result;
- **Severity 3 (SEV3):** An error which drove to the system crashing and/or to the loss of unsaved data.

Keeping this configuration on mind and based on the analysis represented on the figure 7.10, we can verify a decrease of errors made by the subject, from the Question 5 to

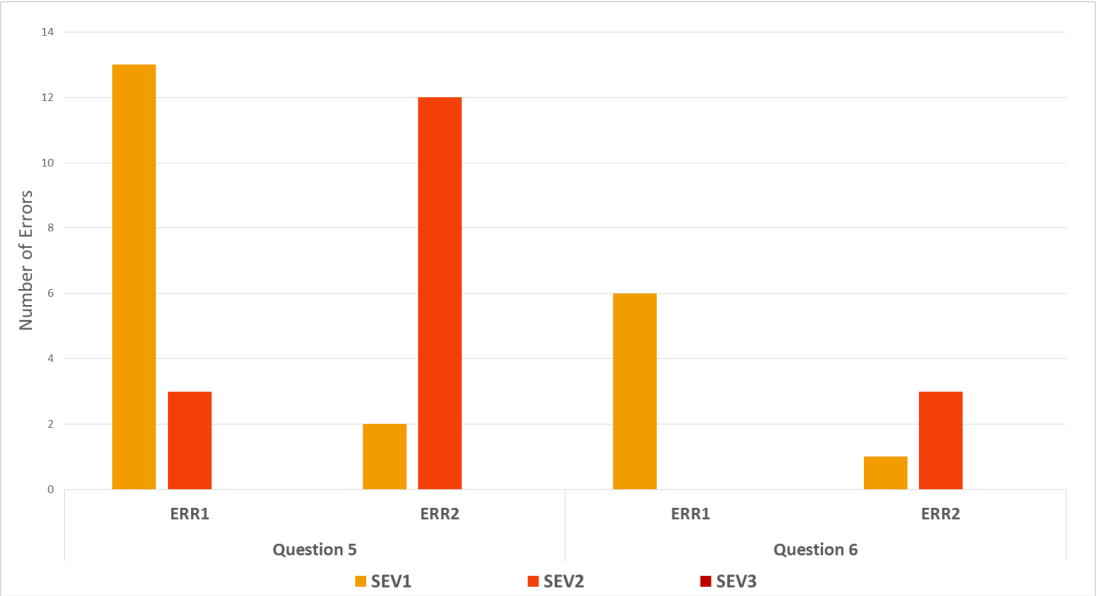


Figure 7.10: Number of Errors by Type and Severity Level

the Question 6. Once again, this can be explained by the simple fact that the subject learns with the errors made on previous experiments and by the fact that the configuration for the Questions 6 is slightly simpler than the Questions 5, namely the nonexistence of parametrization for the HVSC-Y asset.

In fact, when analysing, in detail, the type **ERR2** - Wrong configurations - the majority of mistakes were made on the Question 5, counting a total of 14 occurrences, from which, 12 occurrences were at Severity 2.

Related with the Question 5, it was observed the following specific configuration mistakes:

- **Subjects planning to use the Conditions container:** To, somehow, extract the three regions with more debt. Not recognizing, immediately, the Function container, the contained assets and its respective goals, namely the Top N;
- **Subjects choosing the Bottom N instead of the Top N:** This mistake suggests that some subjects did not understand the goals of the Top N and Bottom N asset;
- **Observed mistakes related with the misconfiguration of the Top N asset:** This mistakes suggests that, despite the user recognized the Function container and the Top N asset, he did not understood how to configure it;
- **Incorrect mapping between the meta-data attribute and the widget property:** For example, the subject choose the Month attribute on the X Axis, instead of Region;
- **Subjects selecting the wrong widget type:** Uniquely driven by the subject distraction when reading the question.

In particular, the misconfiguration of the Top N asset were, in most of the cases, related with the subject forgetting to select the value for the *Ordered By* parameter, as illustrated on the figure 7.11, which led to incorrect or unexpected results, namely the presentation of validation error messages. This mistake in particular, suggests that the user did not understood every aspects of the Top N asset configuration. Namely, the fact that it needs the *Ordered By* parametrization to know on which criteria it should sort the data for classification.

In addition to that, it suggests the existence of problems related with the user interface. Although the system validates the configuration of the asset on backstage and explains the mistakes made, the Editor enables the triggering for back end services even with this invalid configuration. As such, on this specific case, the Editor should, rather, grant that the parameter is required.

As a result, it affected, directly, on the rising of time spent to solve the exercise, on the difficulty expressed by the subjects and, therefore, on their emotional state.

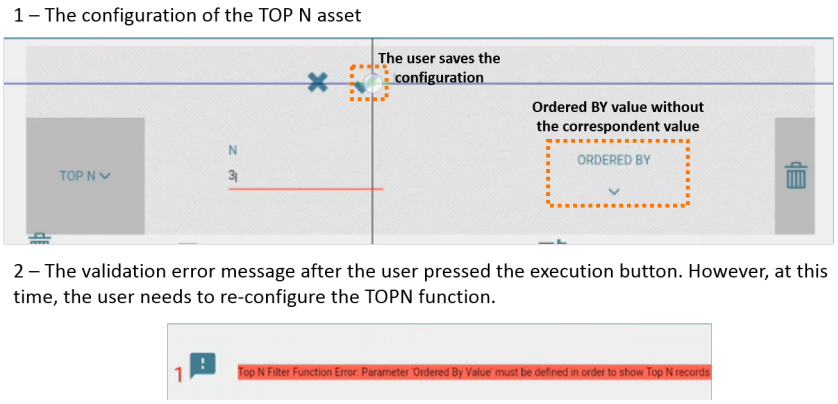


Figure 7.11: Top N Wrong Configuration

Related with the Question 6, it was observed the following mistakes:

1. **Incorrect mapping between the meta-data attribute and the widget property:** For example, the subject choose the Month attribute on the Y Axis, instead of Profit;

2. **Mapping the Year on the Series property of the Line-Chart:** To compare the Profit evolution between the current and the past year.

The second mistake, in particular, aligns with the scenarios where the subject needed to be assisted by the evaluator to reach the correct solution, indicating that some subjects did not recognize the Function container and its HVSC-Y asset as the solution for the problem, because, as the subjects reported, they did not remembered how to access this asset. In one particular case, the subject planned to create a model where he would map the Year attribute on the Series property of the Line Chart, the Month as the X Axis and the Profit for the Y Axis. This solution is, indeed, not incorrect, as the generated widget would present, to the subject, a monthly profit evolution grouped into different years,

wherein he could visualize the current year and the previous. However, the subject did not understood that this solution is not the better choice to answer the formulated question, as it does not provide, in terms of percentage, a summary comparison between the profit on the current year versus on the previous year.

Despite of being at the lowest level of severity, another error with some occurrences, for each question, is the **ERR1** - Incorrect interactions with the interface elements. This evidence suggests the existence of some problems related with the Editor user interface. In fact, the majority of mistakes observed were related with the disposal and dimensions of the elements. Namely, in most of the cases the user did not recognized the main interactive elements, resulting in situations where the subject touched in non interactive elements, expecting some response from the interface, causing some frustration on the subject and some increase of task-completion time but not necessarily in a wrong or unexpected result. This scenario is illustrated on the figure 7.12.

Figure 7.12: Incorrect interaction with the container. In this example the subject touched far away from the button to open the meta-data list. Claiming that, the entire container should be touchable.

Another ERR1 mistake observed, was the cases where the subject pressed the execution button, calling for the widget generation process without completing all the expected steps. This is another indicator that suggests the existence of problems related with the user interface of the Editor. For instance and specially for the Question 5, in some cases, the subjects had correctly configured the Top N asset, however, they pressed the execute button to call the back-end services without saving the configurations, leading to an incorrect generated widget - thus being at Severity 2. This specific problem directly influenced the rising of time spent to reach the correct solution.

7.5.2.3 Analysing the Satisfaction

Lastly, the satisfaction level was measured based on the responses of the closing questionnaire. The chart depicted on the figure 7.13 shows the answers related to which question was the most difficult to execute.

At first sight, the results suggests that, the opinions given by the subjects are quite different in comparison with the insights obtained on the task-completion analysis.

Some of the participants related that, despite of the mistakes made on the Question 5 and the improvements, in terms of time spent, from the Question 5 to the Question 6, they struggled to relate the HVSC-Y asset as the solution for the problem. In other cases, although the subjects identified, immediately, the HVSC-Y as the solution, they struggled to remember how to reach the asset, indicating problems on the Concrete Syntax, namely

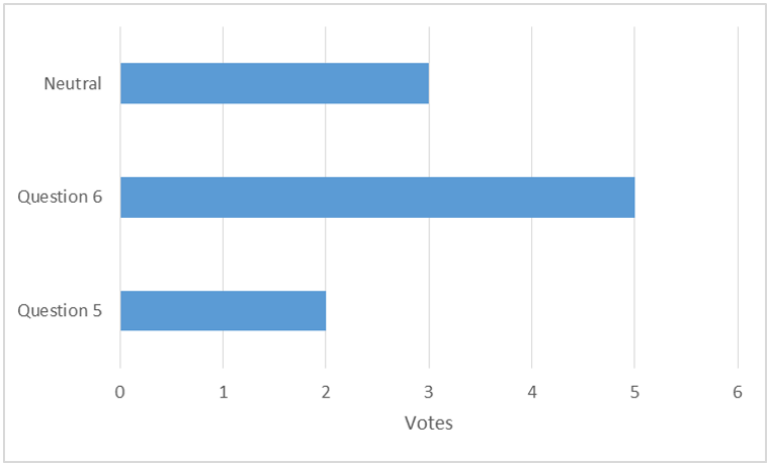


Figure 7.13: Answers regarding the hardest question

related to the lack of suggestiveness regarding the correct meaning of the Functions container.

As such, the common opinions given by the participants were that:

- It was difficult to find the HVSC-Y asset;
- The goal of the Functions container was not very clear;
- In some situations, the Editor needed to provide more feedback;
- Some features of the Editor should have a guiding manual explaining its goals. For example, for each asset of the Functions container, a section with a brief manual.

To close the usability test session we asked the subjects what they think about the usefulness regarding the information monitoring aligned with the user empowerment capability, on mobility contexts. The responses given by the subjects showed us that everyone agrees with the importance of monitoring the data on mobile devices and that the majority agrees on the importance of having the empowerment on the widget's configuration, while on mobility contexts. One subject recognized the usefulness of monitoring on mobility contexts but felt that it was preferable having other people generating its widgets, although they should be fast and efficient.

7.5.3 Threats to validity

The results extracted from the empirical tests enabled us to analyse the usability of the Modelling Language which let us identify points of further improvement. Nonetheless and before we take any conclusions, we need to take into consideration some external factors that are hard to control and can threat the results.

The tests were carried with collaborators of the company, during their office hours, although we strongly encouraged the subjects to participate regarding their availability.

Meaning that, the subjects could be under pressure to rush the usability evaluation. On the other hand, much of the tests were taken during break times, but after work related activities during the morning. Thus, the lack of focus observed in some situations could be, potentially driven by these factors.

The results regarding the subjects satisfaction were extracted from open and closed questionnaires. Some answers given by the subjects are, therefore, subjective and could be affected by emotional and behavior changes, merely by the fact that they were aware of being studied. These factor could, in some cases, justify the scenarios where the subject gave an answer that does not align with the facts observed and analysed.

7.5.4 Conclusions

The results of the conducted tests show the usability of the Modelling Language. In particular, we have based on some usability metrics to evaluate the Modelling Language: Time; Errors and Satisfaction.

Wherein, the time enabled us to provide analysis regarding the efficiency; the errors and task-completion enabled us to evaluate the effectiveness and the satisfaction enabled us to understand their thoughts and perspective regarding the Modelling Language features and the empowerment aspect related to constructing new analytical widgets on mobility contexts.

We can conclude from the conducted tests that:

- The subjects recognized the goal of the prototype, where the majority demonstrate the ability to use the tools to get to the right solution. The results concerning the time and errors from the first to the second practical experiment led us to, in addition, conclude that, over time, every user can learn from past experiences and, therefore, eventually get used with the editor component;
- When it comes to monitoring the business information and be able to configure the widgets, most of the subjects agrees on the user empowerment relevance and recognizes the inherent advantages. However:
  - It is also clear and, in fact, is important to take into account that not every user has the will to be totally empowered. Therefore, it is important to provide different levels of configuration;
  - For this reason, it was conceived the ability to configure the grid dashboard on the target platform. It empowers the end user to configure its dashboard, in terms of position and size of the pre-defined widgets. This feature was, particularly, appreciated by every subject;
- The end users do not pay attention to every detail and feature when they want to see the results as fast as possible. This scenario is likely to occur on mobility contexts, therefore it is very important to decrease noise and simplify the interface

as much as possible, so that the user can achieve their goals with little resources. However:

  - It is also crucial to provide the necessary empowerment features, such that, it can cover every needs and demands. Therefore the trade-off between simplicity and empowerment is an important matter to consider.

7.6 Domain experts validation

On the previous chapter, we analysed the usability of the Modelling Language using a set of subjects representing the end users - The actors who interact with the Modelling Language. On this chapter, we perform the tests with the domain experts with the final goal of collecting useful insights from people specialized in the area and proving if the developed solution fulfills the end user needs.

As mentioned before, on the subjects presentation (section 7.3), we counted with four domain experts, specialized or currently working on the BI domain. From this four subjects:

- One is an experienced consultant of BI. Conducted workshops related to BI tools, participated in several projects, in which, participated in the construction of every layer of the BI architecture - From data modelling, conception of Extraction Transform and Load (ETL) processes to development of operational and analytical solutions including the use of several data visualization tools;
- Two subjects are regular domain experts, who already participated in projects of BI, participated in the conception of data models and ETL processes, used several data visualization tools;
- Other subject is a novice BI consultant who works recently in the BI domain and participated in the conception of data models and ETL processes.

7.6.1 The results

To contextualize the subjects on the matter, they also executed the empirical tests. On the end, a set of closing questions were presented in order to extract their observations and insights regarding the Modelling Language.

As such, the questions were related with:

1. Identifying issues and recommended changes;
2. Classification, in terms of relevance, of a specific set of features;
3. Positive points regarding the solution.

7.6.1.1 The recommended changes

Some changes recommended by the domain experts were related to some aspects concerning the user interface, namely in terms of dimensions, symbols and features. In particular, according to some subjects, the import icon on the Editor it is not very suggestive, because, although it represents well its goal - to import the generated widget - in terms of the end user perspective, it does not make sense to perceive the process as an importing, but rather a saving of the generated widget, as such it should be represented as a diskette icon. In addition, other recommendations were related to the case observed and illustrated in the figure 7.12, wherein the end user subject touched in non-interactive elements, expecting a response from the Editor.

Other recommendations were related to the generated widget, in particular, the widget generated by the HVSC-Y asset. According to some subjects, it is also important that, the widget presents a menu option which merges or divides the two sub-charts. Because, regarding detailed and comparative analysis, where the percentage result is not sufficient, it should be capable of providing the information across different series, where each one is represented by the current and the previous year. For example, for a line chart, it should display a line for each year. However, it is possible to exist widgets where it is not possible or does not make much sense, to join the sub-charts - for instance the pie chart. They also recommended generating a drop down filter incorporated with the generated widget, so that the End-User has, besides, the possibility to select the current year - The previous year would be extracted accordingly with the current year selected.

Some improvements were also recommended in terms of style and design, namely the need of having the ability to control the visual appearance of the widget, namely the background, series colors, font size and the corporate branding.

7.6.1.2 General appreciation

Regarding the features of the Editor, we asked the domain experts to classify the presentation of model validation messages, the conditions container, the functions container and its HVSC-Y asset in particular. The classification about the mapping between the meta data attributes and the widget specific properties was left behind since this aspect is common and well-known in other BI tools.

The results were positive with almost all the domain experts appreciating favorably the presented features, as illustrated in the chart of the figure 7.14.

We also asked about the positive aspects from the Editor. The common responses given by the domain experts were aligned with the advantages inherent with the user empowerment based on the MDD and SPL concepts, thus facilitating the easy and quick configuration and generation of analytical widgets which are scoped to the business requirements.

They also appreciated positively the target-platform, referencing as well that it provides some distinctive features which fits well on mobility contexts, namely the capability

of customize the dashboard, on run-time, based on a grid system.

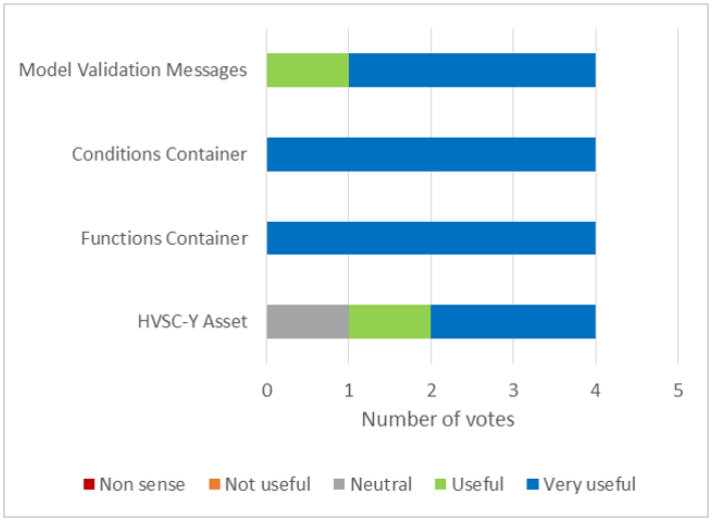


Figure 7.14: Appreciation regarding the Editor features

7.7 Summary and discussion

In this chapter we covered the validation of the prototype. The goal was to determine the extent to which it can be used to empower the end users.

Therefore, based on the empirical techniques of human-machine interaction we evaluated the modeling language by defining an evaluation process which would encompass the analysis of effectiveness, efficiency and satisfaction dimensions with metrics of time, task-completion, errors and with open-ended questions. Simultaneously, two groups were defined based on preliminary interviews with each subject: the end users and the domain experts groups.

The observations and results extracted from the end users tests showed, overall, that they learned and were able to use the Editor to solve the tasks, therefore we can conclude that, in fact, the prototype can be used to empower the end users. Nonetheless and along with their satisfaction, some negative aspects were highlighted, regarding the Editor's interactivity model and the language concrete syntax, which could be addressed on future extensions of the prototype. Perhaps this negative aspects contributed to one particular answer, referring that the subject would prefer having people building the components rather than using the Editor and its modeling language. Despite of that, it is clear that not everyone has the inclination or wants to be fully empowered, as such, it is important to deliver different empowerment levels.

The extent version of the Target Platform provides the capability of building the dashboard with a grid system by drag and drop actions over the widgets - a particular feature appreciated by the subjects. This feature could be taken advantage of, in the situations where the end user does not want to be fully empowered and, as such, does not want to

access the Editor and its modeling language to configure new widgets. To that end, the Target Platform would provide a list of widgets served as pre-defined components that would be picked by the end users to define the dashboard presentation.

The results of the tests executed by the domain experts let us finalize the evaluation process by enabling the gathering of insights from people who work in the BI area, accordingly with the principles of this domain. We extracted their appreciation regarding some specific features along with further points of improvement and extension on the prototype. Giving that they are people who work in the field and have experience in projects with real clients, their validation and suggestions on this prototype are relevant to take into account.

8

Conclusions and Future Work

8.1 Conclusions

On this dissertation, we tackle the need for the user empowerment in the Business Intelligence domain. This area is well-known by companies as the support for decision-making activities. Specifically, we have approached the issue related to the configuration and generation of analytical components for data monitoring on mobile gadgets, particularly on tablet devices. Based on the problem statement we defined a research question of whether it was possible to empower the users, so that, a cooperative system was established.

In this sense, we conceived a prototype which, based on Software Product Line concepts, delivers a system of cooperation between two main groups - the domain experts and the end users. The domain experts are, essentially the product line developers who have the control and the responsibility to supply the production assets, scoped to the business requirements. The end users group represents the business users, who want to monitor the data and need to be empowered so that they can develop and generate its own products. They are, essentially, the product developers who picks the supplied assets and configure them to generate the desired product.

This generative capability is granted by the Model-Driven concepts and the standard technologies, which forms the core of the prototype. Based on the model instance, configured by the end user on the modeling language of the editor, the widget code is generated, being compatible with the target platform which presents the corporate information for monitoring purposes.

The solution's main result is the empowerment of the end users so that they can configure its widgets and the dashboard, based on their preferences and demands, without

the need of being a BI specialist.

Besides, it optimizes the domain experts development processes. In fact, with this prototype, despite the production line assets being aligned with a defined scope based on business requirements, its portability is still leveraged because the assets are generic and horizontal to the business concepts. Meaning that, in the best cases, with the emergence of new clients, the production line assets developed for other contexts can also be allocated towards the new customer's needs. This strategy is still aligned with the initial motivation of the enterprise: To have a customizable and extensible solution where they have full control regarding which components are delivered to the clients. However, the development efforts are now optimized, towards the conception of generic assets, horizontal to the business concepts.

Lastly, we conducted a usability test so that we could evaluate if the editor and its modeling language are suitable to be used by different types of end users. In this sense we collected a set of subjects, interviewed them and identified two groups: the end users and domain experts. To obtain insights regarding the editor and the modelling language, we extracted the time, errors and task-completion rates from the end users experiments, to analyse the efficiency and effectiveness. From the domain experts, we took advantage of their knowledge to validate the prototype, regarding best practices on BI, as well as to gather useful insights and recommendations which can also serve as further points of extension. The results led us to conclude, that, based on the number of successful tasks, the editor and its modelling language has the potential to be used by end users to fulfill their needs. But, it is important to note that, not everyone wants to be entirely empowered and, typically on this cases, they just prefer having the ability to configure the presentation aspects from the grid dashboard, based on the pre-defined widgets presented on the widget repository of the target platform.

8.2 Future Work

Besides the prototype developed and the usability evaluation delineated, which, based on the facts observed, the end users opinions, the domain experts recommendations, enabled us to gather further points of improvement, it was, also, opened the opportunity to approach further challenges.

Namely, it is interesting to develop additional user interfaces for the editor component, and possibly other concrete syntaxes. Along with some improvements suggested by the end users and recommended by the domain experts. Next, conduct a usability evaluation with the goal of defining comparative analysis along with the results extracted on this dissertation.

Another interesting improvement for the editor would be the extension of the validation messages so that they point to or emphasize which model element is incorrect.

For this prototype, we have already deployed a set of generic purpose assets. Naturally, with the system maturation, further assets can be deployed and additional features

incorporated namely, widgets based on other chart types, widgets for geo-referential data monitoring, event-based generic assets - for instance, notifications - which can be used to customize the presentation of a specific widget based on a given event, widgets with drilling actions.

The editor and the target platform were conceived to be optimal in mobile tablet devices on landscape orientation. Further efforts are needed in order to optimize the interface in smaller devices, namely smart phones.

It is also important to take into account that, this prototype is a proof of concept. As such, for production contexts, it is important to explore other topics which were not directly approached on this dissertation. Namely, the topics related to security, authorization and access control, scalability concerns, corporate branding.

Bibliography

[A B12a] A. Barišić, Vasco Amaral, Miguel Goulão. "Usability Evaluation of Domain-Specific Languages". In: *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (Quatic 2012)* (September 3 2012). DOI: 10.1109/QUATIC.2012.63.

[A B12b] A. Barišić, Vasco Amaral, Miguel Goulão and Bruno Barroca. "Evaluating the Usability of Domain-Specific Languages". In: *Marjan Mernik,"Formal and Practical Aspects of Domain-Specific Languages: Recent Developments IGI Global,2013"* (9 Ago 2012).

[AH10] D. Airinei and D. Homocianu. "The Mobile Business Intelligence Challenge". In: *Economy Informatics* 10.1 (2010), pp. 5–12. URL: goo.gl/HCdMe0 (visited on 01/15/2015).

[CZL06] L. Cao, C. Zhang, and J. Liu. "Ontology-based integration of business intelligence". In: *Web Intelligence and Agent Systems* 4.3 (2006), pp. 313–325. ISSN: 15701263.

[Cha+13] L. K. Chan, H. K. Tan, P. Y. Lau, and W. Yeoh. "State-of-the-Art Review and Critical Success Factors for Mobile Business Intelligence". In: *Communications of the IBIMA* 1.2013 (2013), pp. 708–717. ISSN: 19437765. DOI: 10.5171/2013.246123.

[Eck11] W. Eckerson. "A Complete Primer To Mobile BI". In: *BeyeNetwork* (2011), pp. 1–16.

[Eck13] W. Eckerson. *The Promise of Self-Service BI*. 2013. URL: goo.gl/2mypsq (visited on 01/15/2016).

[Ele11] C. Elena. "Business intelligence". In: *Journal of Knowledge Management, Economics and Information Technology* 1 (2011), p. 101. ISSN: 1050-9135. arXiv: z0037.

[Fow10] M. Fowler. *Domain-Specific Languages*. Vol. 5658. Addison-Wesley Professional, 2010, p. 640. ISBN: 9780321712943. DOI: 10.1007/978-3-642-03034-5.

[FHK09] U. Frank, D. Heise, and H. Kattenstroth. "Use of a Domain Specific Modeling Language for Realizing Versatile Dashboards". In: *In Proc. of the 9th OOPSLA workshop on domain-specific modeling*. 2009.

[FHH00] E. Frøkjær, M. Hertzum, and K. Hornbæk. "Measuring Usability : Are Effectiveness , Efficiency , and Satisfaction Really Correlated ?" In: *ACM CHI 2000 Conference on Human Factors in Computing Systems* 2.1 (2000), pp. 345–352. DOI: 10.1145/332040.332455.

[GS03] J. Greenfield and K. Short. "Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools". In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. OOPSLA '03. Anaheim, CA, USA: ACM, 2003, pp. 16–27. ISBN: 1-58113-751-6. DOI: 10.1145/949344.949348. URL: http://doi.acm.org/10.1145/949344.949348.

[Gre+04] J. Greenfield, K. Short, S. C. Cook, S. C. Kent, and J. Crupi. *Software factories : assembling applications with patterns, models, frameworks, and tools*. Indianapolis (Ind.): Wiley, 2004. ISBN: 0-471-20284-3.

[HM07] P. Hanrahan and J. Mackinlay. "Visual Analysis for Everyone". In: *Tableau Software* (2007), p. 27. URL: goo.gl/63KwZh (visited on 12/28/2012).

[Her] P. Hernandez. *Reports vs Dashboards*. URL: https://goo.gl/hZoLrg (visited on 12/12/2015).

[IW11] C. Imhoff and C. White. "Self-Service Business Intelligence: Empowering Users to Generate Insights". In: *TDWI Report* (2011), pp. 1–37. URL: http://www.sas.com/resources/asset/TDWI%7B%5C_%7DBestPractices.pdf (visited on 02/01/2016).

[Kau10] A. Kaushik. *Web Analytics 2.0*. Wiley, 2010, pp. 1–447. ISBN: 9780470529393.

[KT08] S. Kelly and J. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008. ISBN: 9780470249253. URL: https://books.google.pt/books?id=GFFtRFkuU%5C_AC.

[KR02] R. Kimball and M. Ross. *The data warehouse toolkit: the complete guide to dimensional modelling*. 2002, pp. 1–447. ISBN: 0471200247. DOI: 10.1145/945721.945741.

[Küh06] T. Kühne. "Matters of (Meta-)Modeling". In: *Software and System Modeling* 5.4 (2006), pp. 369–385. DOI: 10.1007/s10270-006-0017-9.

[Luh58] H. Luhn. "A Business Intelligence System". In: *IBM Journal of Research and Development* 2.4 (1958), pp. 314–319. ISSN: 0018-8646. DOI: 10.1147/rd.24.0314. URL: http://altaplana.com/ibmrd0204H.pdf (visited on 11/15/2015).

[MPM15] R. Mcshane, L. Product, and M. Manager. "Three Paradigm Shifts Regarding Mobile Business Intelligence". In: (2015).

[Min12] I. Minar. *MVC vs MVVM vs MVP*. 2012. URL: https://plus.google.com/+AngularJS/posts/aZNVhj355G2 (visited on 03/04/2016).

[OE15] M. K. Oksman and Eugene. *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options*. 2015. URL: goo.gl/hkkOYw (visited on 02/03/2016).

[Ped10] Pedro Gabriel, Miguel Goulão and Vasco Amaral. "Do Software Languages Engineers Evaluate their Languages?" In: *Proceedings of the XIII Congresso Iberoamericano en "Software Engineering" (CIbSE'2010), Universidade del Azuay* (Abril 2010).

[Pow] D. J. Power. *Data-Driven DSS Resources*. URL: http://dssresources.com/dsstypes/ddss.html (visited on 12/10/2015).

[SM03] T. Schadler and J. C. McCarthy. *Mobile is the New Face of Engagement*. 2003. URL: media.cms.bmc.com/documents/Mobile_Is_The_New_Face_Of.pdf (visited on 12/29/2015).

[Sch] K. Schlegel. *How to Deliver Self-Service Business Intelligence*. URL: goo.gl/XwDkOw (visited on 11/18/2015).

[VS13] K. I. M. Verkooij and M. Spruit. "Mobile Business Intelligence: Key Considerations for Implementation Projects". In: *Journal of Computer Information Systems* Fall 2013 (2013), pp. 23–33.

[Voe+13] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. Kats, E. Visser, G. Wachsmuth, S. Benz, and B. Engelmann. *DSL Engineering*. dslbook.org, 2013, p. 560.

Additional Information

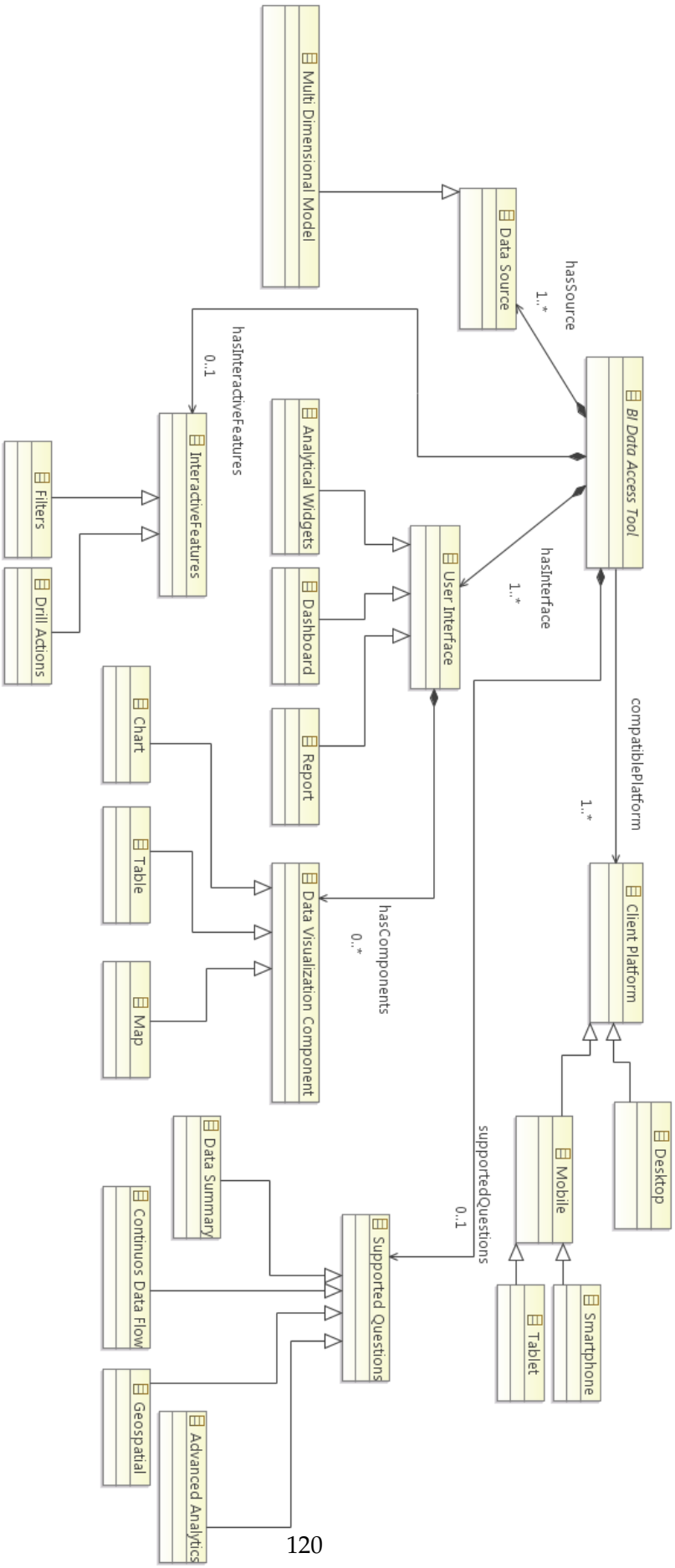


Figure A.1: BI Concepts Model

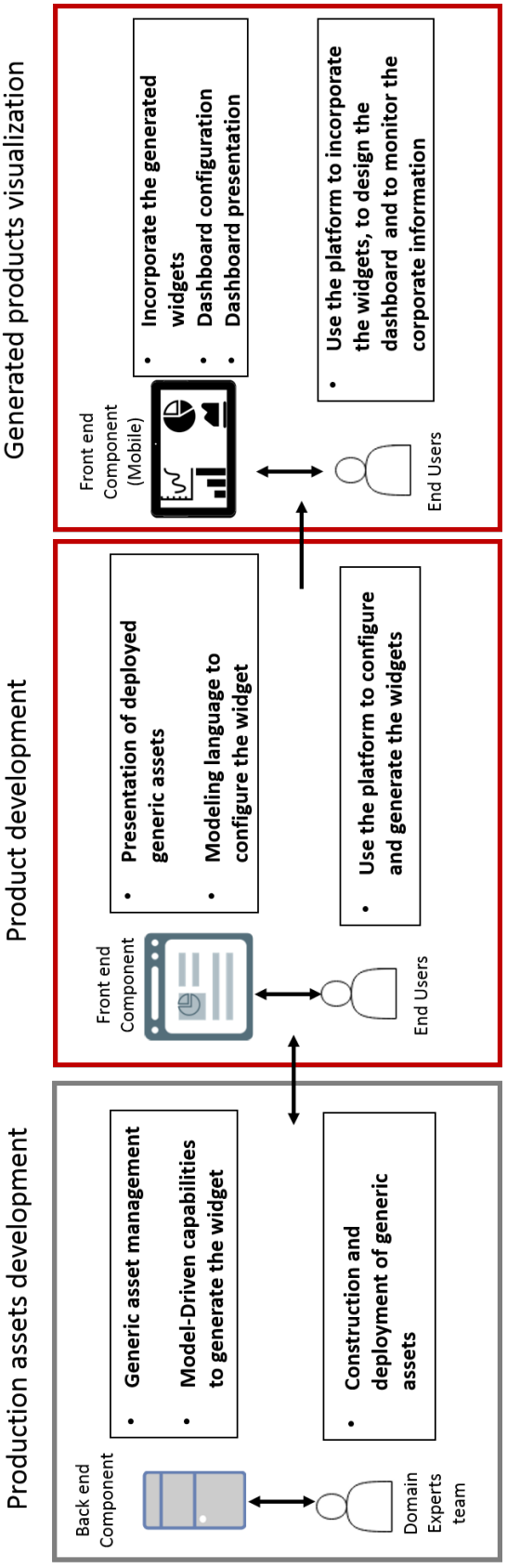


Figure A.2: Solution overview

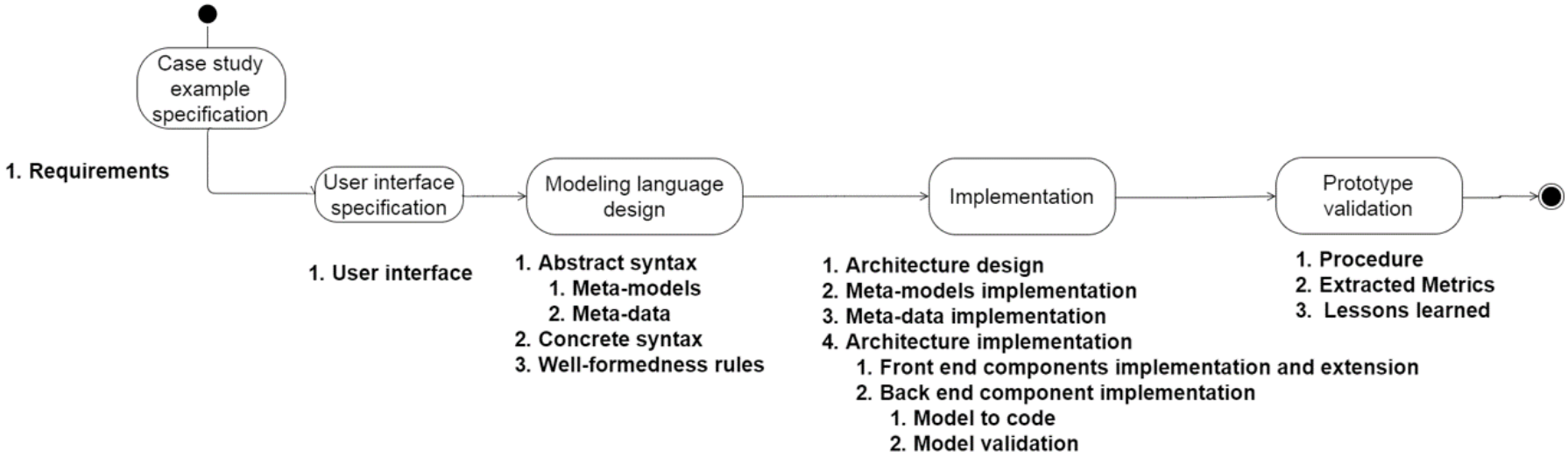
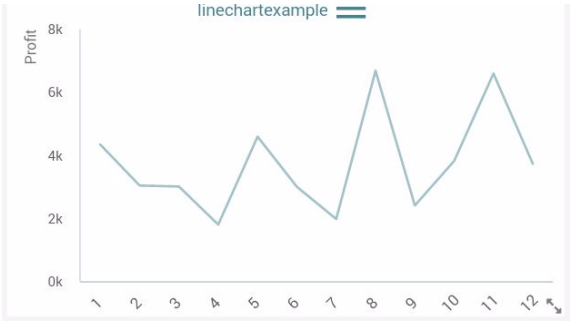
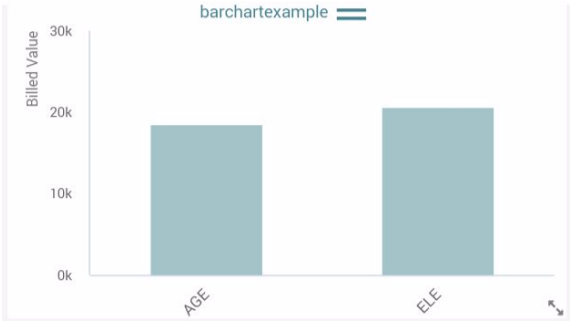


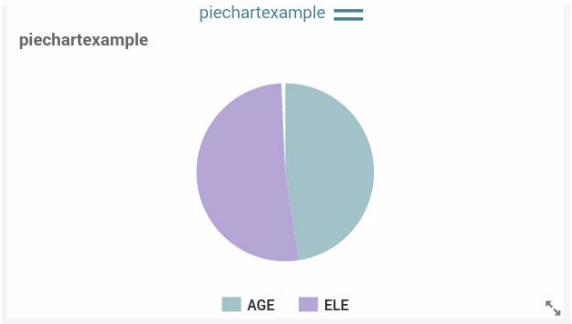
Figure A.3: Conception stages of the prototype



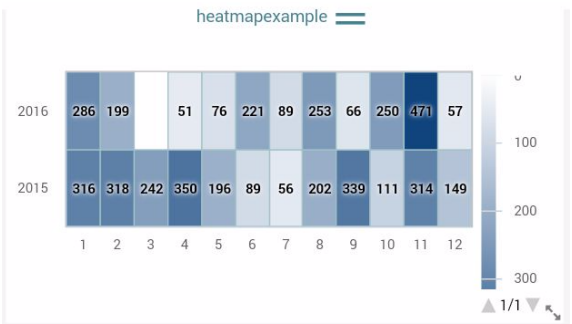
(a) Line Chart



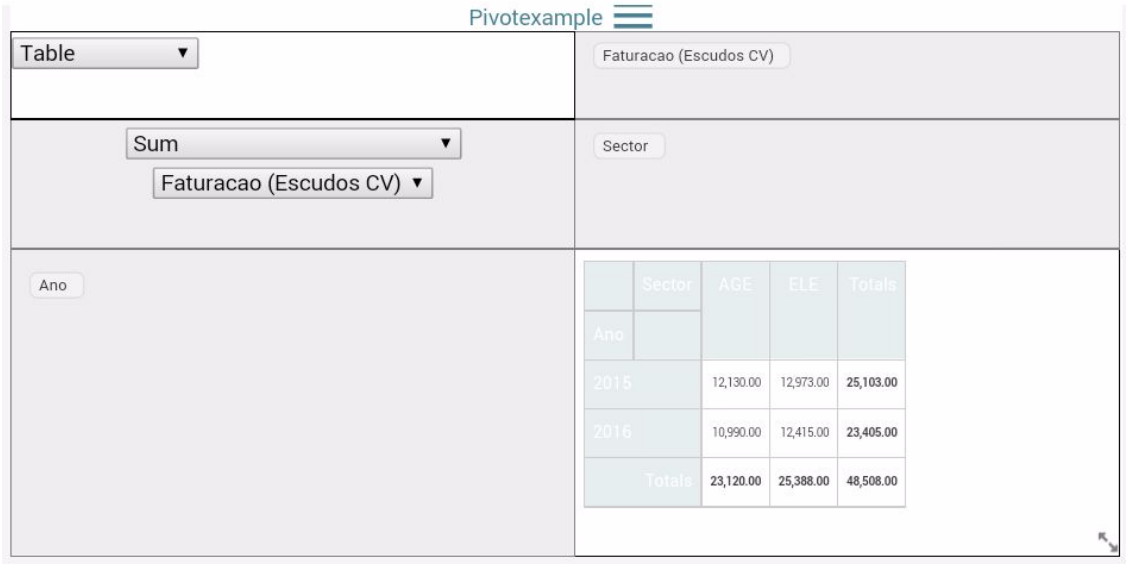
(b) Bar Chart



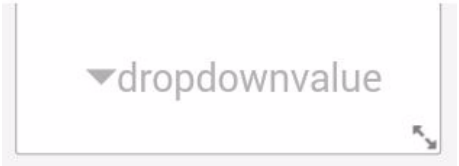
(c) Pie Chart



(d) Heat Map



(e) Pivot Table



(f) Drop Down Filter

Figure A.4: Widget Examples

B

Usability Tests: Additional Material

B.1 End user closing questionnaire

1. Which experiment you found the most difficult to perform?

   _____

2. What were the reasons that hindered you to solve the experience?

   _____
   _____
   _____
   _____

3. As a business member of an organization, do you consider useful having the ability to generate widgets that demonstrate the performance of your business

   No. I would rather have other people to generate the widgets for me. ☐

   I find it useful ☐

4. Do you consider useful to analyze business information on mobile devices, particularly in Tablets?

   Yes ☐ No ☐

B.2 Domain expert closing questionnaire

1. Do you recommend any changes on the editor, including the modeling language? If so tell us what

_____
_____
_____

2. Do you recommend changes regarding the widgets? If so tell us what

_____
_____
_____

3. Do you recommend any other changes? For example the target platform? If so tell us what

_____
_____
_____

4. What are the positive aspects of the presented prototype?

_____
_____
_____

5. Please rate the following features in terms of usefulness (rate 0 to 5):

   1 – Non sense 2 – Not useful 3 – Neutral 4– Useful 5 –Very useful

   a. Container *Functions*: ____
   b. *Homologus* VS *Current Period* Asset: ____
   c. Container *Conditions*:____
   d. Model validation and messages presentation: _____

6. Do you think the editor is suitable for a business user?

_____

B.3 Contextual information

Before you start, please read carefully this contextual information. If you have any question please feel free to ask. Thank you!

- The app you are going to test is a generator for analytical widgets;
- The page that you can see at the moment is the Editor. Wherein, you can configure the *widgets*, mapping components denominated as ***blocks*** on specific properties represented as ***containers***;
- Every widget has its specific ***containers***. For example, a Line Chart has:
  - X Axis: Container representing the horizontal axis;
  - Y Axis: Container representing the vertical axis;
  - Series: Container representing the lines drawn on the chart;
- Common to all widgets, the container ***Conditions*** enables you to add conditions with the goal of filtering the information;
- Also common to all widgets, the container ***Functions*** has a set of generic **tools/functions/scripts** which can be used to help you respond certain type of questions;
- The goal is to map the **blocks** on the aforementioned **containers.** The **blocks** represents the information you have on your data repository, the functions or conditions you instantiate on the editor:
  - The data repository blocks are represented by two type of icons:

     If it is **descriptive** (for example: Product Name)

     If it is a **measure** (for example: Price)
  - The **conditions** are represented by  ;
  - The **functions** are represented by  ;
- Every container knows which type of blocks it should accept. For example, the container Y Axis of the Line Chart, only accepts  the type . The type of blocks acceptable, for each container, is represented by the respective icon on the top right corner;
- When pressed, the icon  triggers processes that generates the widget;
- If you make some misconfigurations during the test, the widget generator will invalidate the configurations you have submitted, by presenting you error or warning messages explaining the situation;
- The generated *widgets* can be saved by tapping the icon  ;
- The widgets that were saved can be selected on the page Widget Repository. To go there, you must exit the Editor page. Then you can choose the widgets you want to compose your dashboard;