Pontifícia Universidade Católica
do Rio de Janeiro

Luciane Calixto de Araújo

Model Driven Questionnaires based on a Domain Specific Language

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marco Antônio Casanova

Rio de Janeiro

October 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Luciane Calixto de Araujo

Model Driven Questionnaires based on a Domain Specific Language

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

Prof. Marco Antonio Casanova
Advisor
Departamento de Informática – PUC-Rio

Prof. Antônio Luz Furtado
Departamento de Informática – PUC-Rio

Prof. Luiz André Portes Paes Leme
Departamento de Ciência da Computação - UFF

Rio de Janeiro, 03 October, 2019

Luciane Calixto de Araujo

The author graduated in Control and Automation Engineering from Federal University of Santa Catarina (UFSC), Florianópolis – Brasil in 2007. She joined the Graduate Program in Informatics at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in 2016.

Bibliographic data

Araujo, Luciane Calixto de

Model Driven Questionnaires based on a Domain Specific Language / Luciane Calixto de Araujo; advisor: Marco Antonio Casanova. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 135f. : il. ; 29,7 cm

1. Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Busca por palavras-chave. 3. DSLs. 4. Pesquisas estatísticas. 5. MDE. 6. Questionários I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CCD: 004

To Willian, whose love, patience and support brought me here.
To mom and dad, who gave me the tools necessary to an academic pursuit.
To Malu, my little girl and my biggest privilege.

Acknowledgements

Right after I stared the road of this research project I was blessed with my daughter, Malu de Araujo Silva. It has been a huge challenge to balance motherhood, my work at IBGE and this research. Still, with the support of my partner, Willian Alves da Silva, my parents, Telma Calixto de Araujo and Edélcio Tavares de Araujo, and my family, I'm finally reaching the finish line.

I also wouldn't be here without the unending patience, understanding and guidance of Professor Marco Antonio Casanova. For that, my special thank you and admiration. His tutoring was imperative in choosing this research topic which ended being an opportunity to acquire new and exciting knowledge and to mitigate multiplexing between research and work.

My coworkers at IBGE contributions were invaluable. I will not cite names to avoid being unfair. But this dissertation could not have been done without the shared experiences and afternoon conversations while eating some cake and coffee. Thank you all!

Thank you PUC-Rio and CNPq for providing the financial means that allowed me to be part of the Programa de Pós-graduação em Informática.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance code 001.

To all my classmates, professors and staff from the PUC-Rio Department of Informatics, thank you for always being ready to listen and accommodate!

Abstract

Araujo, Luciane Calixto de; Casanova, Marco Antonio (Advisor). **Model Driven Questionnaires based on a Domain Specific Language**. Rio de Janeiro, 2019. 135p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Surveys are pervasive in the modern world with its usage ranging from the field of customer satisfaction measurement to global economic trends tracking. At the core of survey processes is data collection which is, usually, computer aided. The development of data collection software involves the codification of questionnaires which vary from simple straightforward questions to complex questionnaires in which validations, derived data calculus, triggers used to guarantee consistency and dynamically created objects of interest are the rule. The questionnaire specification is part of what is called survey metadata and is a key factor for collected data and survey quality. Survey metadata establishes most of the requirements for survey support systems including data collection software. As the survey process is executed, those requirements need to be translated, coded and deployed in a sequence of activities that demands strategies for being efficient and effective. Model Driven Engineering enters this picture with the concept of software crafted directly from models. In this context, this dissertation proposes the usage of a Domain Specific Language (DSL) for modeling questionnaires, presents a prototype and evaluates DSL as a strategy to reduce the gap between survey domain experts and software developers, improve reuse, eliminate redundancy and minimize rework.

Keywords

model driven engineering; domain specific languages; survey questionnaires; data collection, statistical surveys

Resumo

Araujo, Luciane Calixto de; Casanova, Marco Antonio (Orientador). **Questionários orientados por modelos baseados em DSL**. Rio de Janeiro, 2019. 135p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Pesquisas são pervasivas no mundo moderno e seu uso vai de medidas de satisfação de consumidores ao rastreamento de tendências econômicas globais. No centro do processo de pesquisa está a coleta de dados que é, usualmente, assistida por computador. O desenvolvimento de software destinado à coleta de dados em pesquisas envolve a codificação de questionários que variam de simples sequências de questões abertas à questionários complexos nos quais validações, cálculo de dados derivados, gatilhos para garantia de consistência e objetos de interesse criados dinamicamente são a regra. A especificação do questionário é parte dos metadados da pesquisa e é um fator chave na garantia da qualidade dos dados coletados e dos resultados atingidos por uma pesquisa. São os metadados da pesquisa que estabelecem a maior parte dos requisitos para os sistemas de suporte a pesquisas, incluindo requisitos para o software de coleta de dados. À medida que a pesquisa é planejada e executada, esses requisitos devem ser compreendidos, comunicados, codificados e implantados, numa sequência de atividades que demanda técnicas adequadas para que a pesquisa seja eficaz e efetiva. A Engenharia Orientada a Modelos (Model Driven Engineering) propõe estratégias que visam alcançar esse objetivo. Neste contexto, esta dissertação propõe o uso de Linguagens de Domínio Específico (Domain-specific Languages - DSLs) para modelar questionários, apresenta um protótipo e avalia DSLs como uma técnica para diminuir a distância entre especialistas de domínio e desenvolvedores de software, incentivar o reuso, eliminar a redundância e minimizar o retrabalho.

Palavras-chave

engenharia orientada a modelos; linguagens de domínio específico; questionários de pesquisas; coleta de dados; pesquisas estatísticas

Table of contents

List of figures

List of tables

List of code samples

Abbreviations

AST	Abstract Syntax Tree
CAPI	Computer-assisted person interviewing
CATI	Computer-assisted telephone interviewing
CSPA	Common Statistical Production Architecture
DSL	Domain-specific Language
DSML	Domain-specific Modeling Language
DBMS	Database Management System
DDI	Data Documentation Initiative
IBGE	Brazilian Institute of Geography and Statistics
GPL	General Purpose Language
GSBPM	Generic Statistical Business Process Model
GSIM	Generic Statistical Information Model
M2T	Model to Text
M2M	Model to Model
MBT	Model Based Testing
MDA	Model-driven Architecture
MDD	Model-driven Development
MDE	Model-driven Engineering
MDSD	Model-driven Software Development
MDSE	Model-driven Software Engineering
MPS	Meta Programming System
OO	Object Oriented
PeNSE	National Survey of Students Health
PNAD	Continuous National Household Survey by Sampling
PNS	National Health Survey
POF	Family Budget Survey
SDMX	Statistical Data and Metada eXchange
SDK	Standard Development Kit
SQL	Structured Query Language
UNECE	United Nations Economic Commission for Europe

1
Introduction

1.1. Motivation

In this work, the term survey refers to clearly defined methods for gathering data about specific entities with the goal of understanding a phenomenon. Among the many approaches for surveys data collection, it is the usage of structured questionnaires that are applied to a population or to a sample of a population. This dissertation uses the term questionnaire-based survey domain to indicate surveys executed using data collection strategies implemented using questionnaires.

Survey methodologies are adopted by all knowledge fields, from engineering to social sciences. As such, the importance of surveys cannot be overstated. A relatively new research technique, surveys have evolved from informal unstructured presential interviews to highly complex questionnaires, that are extensively tested and paired with strict interview rules.

Survey methodology is the product of multidisciplinary efforts to identify principles for the design, collection, processing and analysis of data. With roots in the mathematical, social and computer sciences, at a first look it seems that survey methodology is a well-defined knowledge field. But that couldn’t be further from the truth. Survey methodology information is scattered and, what on the surface seems quite simple and straightforward, hides a large amount of complexity.

As any other area of knowledge surveys have been tremendously influenced by information technology advances in the XX Century. Computer technology saw processor, storage and memory capacity and performance go through a series of improvements. At the same time, the creation of the Internet and mobile technology have dramatically changed the way surveys are conducted. Nowadays, large scale surveys rely heavily on information technology to guarantee results with adequate quality at reasonable costs.

At first sight, developing software for survey data collection seems to be an ordinary software engineering task. After all, questionnaires are forms for which a large number of different solutions and development strategies exists. That might

be the case for small surveys or for surveys based on simple questionnaires. Nonetheless, when analyzing large scale statistical production processes, surveys can easily become highly complex, with its questionnaires defining intricate data structures that have specific behaviors associated.

In engineering, complexity is frequently handled by raising abstraction levels and it is not different in Software Engineering. Creating better abstractions has been a constant since the first computer language was invented. Model-driven Software Engineering (MDSE) aims at raising computer language abstraction further by making models first class citizens. Closely related to reuse engineering and domain engineering, the ideas behind MDSE are not new. It comes in many flavors, which makes using it a challenging task. Research is still looking to understand how to choose the best MDSE approach to a specific problem.

Among the possible approaches for applying MDSE, it is the usage of DSLs as tools to create models that can be used in different ways during software development. Would applying MDSE and, more specifically, DSLs help taming the complexity of survey related software? What are the benefits? What are the disadvantages? How can MDSE be used for improving software engineering processes and quality in questionnaire-based survey domain?

1.2. Goal and contributions

This dissertation main goal is to understand the problems and advantages that using MDSE on questionnaire-based surveys domain entails. On pursuing that goal, the following objectives will be fulfilled:

- To provide a clear view of the state of the art in applying MDSE as a mean to improve software development in the questionnaire-based surveys domain;
- To prototype a DSL for modeling questionnaires called SLang;
- To evaluate the usage of a DSL for questionnaire modeling by adopting the prototyped DSL in a practical scenario.

1.3. Dissertation structure

The present work is organized following DSL development phases. Chapter 2 discusses questionnaire-based surveys domain, presents concepts and terminology for understanding MDSE in the context of this research and describes the theoretical background for the decisions made while prototyping SLang. Chapter 3 focuses on analyzing and describing the statistical survey questionnaires domain. Chapter 4 presents aspects related to the design and implementation of SLang. Chapter 5 reports on a practical usage of SLang by using it in the context SInterviewer, a data collection software. Chapter 6 presents the conclusions of this research and potential future developments. It is important to point out that although this dissertation is organized sequentially, developing SLang and SInterviewer was an interactive and incremental process.

2
Why a DSL-based questionnaire?

2.1. Introduction

Survey methodology is a multidisciplinary field that has its roots on the mathematical, social and computer sciences. It seeks to identify principles about the design, collection, processing and analysis of data that are linked to the cost and quality of survey estimates. It is the survey methodology that gives us tools to reap the benefits of research using surveys. One could think that, given its importance, survey methodology literature would be consistent and well organized. But that is not the case. As a matter of fact, it is widely scattered (GROVES, FOWLER JR., *et al.*, 2009). That fact might pose a challenge to those trying to form an overall picture of surveys and, more specifically, of survey questionnaires. With that goal in mind, this work first step is to explain what a survey is, from the point of view of those creating software to support surveys. A starting point in understanding this field is to have a formal definition of survey.

A survey is a systematic method to gather data about (a sample of) entities with the purpose of constructing quantitative descriptors for the attributes of a larger population from which the entities are members. Quantitative descriptors are called *statistics* and represent quantitative summaries of observations on a set of elements (GROVES, FOWLER JR., *et al.*, 2009). As such, surveys are pervasive in the modern world with its usage ranging from the field of customer satisfaction measurement to global economic trends tracking. It is the research method most commonly used to understand how societies work and to test theories of behavior in social sciences research. In fact, most areas of knowledge, make use of surveys: governments monthly release data on unemployment and inflation; economists and policy makers are constantly relying on surveys to make informed decisions; doctors use surveys as a methodology in studies and trials that map diseases and health issues; engineers are constantly gathering data to certify quality and reliability of products. (MOORE, MCCABE and CRAIG, 2009).

From the above definition, systematic data collection is at the core of what a survey is. Systematic data collection can be done in many ways. Surveys can use administrative records, interviews, instrument-based measurements, among many other data collection strategies. Still, the usage of a questionnaire is by far the most common data collection strategy (SARIS and GALHOFER, 2014). In questionnaire-based surveys, it is the questionnaire that gives meaning to the collected data. It is through the questionnaire questions that data can be understood and becomes information.

In the early days of surveys, little attention was given to how to formalize a questionnaire, how to conduct a questionnaire-based interview and how to word questions. Interview organization and execution would lie on the interviewer's perspective and experience. An interviewer would receive a list of objectives, such as age, occupation, education and from that list would choose how to conduct the interview. But surveys usage grew fueled by the evolution of methods that made data collection progressively quicker and cheaper (GROVES, FOWLER JR., *et al.*, 2009). As the demand for surveys grew, the impact of careful planning and clear definitions changed the way surveys were done.

One of the forces behind this expansion was the evolution of survey methodologies and techniques. The other was technology. It was technology that made viable the migration from face-to-face interviews to mail in first half of the twenty century. Telephone interviews became popular between the 60's and the 90's using Computer-assisted Telephone Interviewing (CATI), followed, more recently, by the use of Internet and mobile technology, which gave birth to Computer-assisted Personal Interviewing (CAPI). Each new technology enhanced and extended the range of opportunities and possibilities for survey researchers as well as introduced new challenges and issues (GROVES, 2011).

In 2005, five trends were pointed in the universe of surveys: (1) the move from interviewer-administered to self-administered surveys; (2) the move from verbal (written or spoken) inputs and outputs to visual and haptic or sensorimotor inputs and outputs; (3) the move from fixed to mobile information and communication technology, for data collectors and for respondents; (4) the move from discrete surveys to continuous measurement; and (5) the move from data only,

to data, metadata and paradata[1]. The implications of those trends, as they progressively took place in the universe of surveys, are the multiplication of survey data collection modes, survey data collection process democratization, an increased specialization due to the proliferation of new methods, a higher demand for communication between areas of expertise along the statistical survey production chain, among others (COUPER, 2005). As of 2019, most of these trends are no longer trends, but became common practice demanding responses for each of the challenges imposed by technology evolution and its impact on survey practices.

Right at the center of those challenges is the data collection software. Data collection software enables self-administered surveys, allows the usage of new input types and evolves constantly to support new methods and medias. Data collection software also has to be available in a multi-platform environment and provide tools that enable continuous measurement while collecting paradata.

It is in this context, that the concept of a DSL-based questionnaire comes alive. This chapter focuses on describing the reality surrounding questionnaire-based survey software. Section 2.2 discusses what is hard about questionnaires and statistical survey data collection software, followed by an overview of tools available for statistical survey data collection in section 2.3. In section 2.4 the main concepts involved in model driven software development are presented. Finally, section 2.5 contains the statement of our research problem and goals.

2.2. What is difficult about survey questionnaires?

The present work focuses on questionnaire-based surveys in which the questionnaire plays a central whole as the survey measurement instrument. Questionnaire-based surveys are the most common approach for research in the social, economic and behavioral fields. For example, the number of articles and publications in the field of sociology, in which research was done using a questionnaire to collect data, increased from 24.1% in 1949-1950 to 69.7% in 1994-1995. In the field of economics the increase was of 5.7% to 42.3% in the same period (SARIS and GALHOFER, 2014).

[1] In the context of statistical surveys metadata is data about the collected data and paradata is data about the data collection process (Couper, 1998).

A superficial look, might lead to the wrong conclusion that a survey questionnaire is just set of questions, displayed as a form to be filled up. Sometimes that might be true. It is even fair to say that most questionnaires are, in fact, simple, with few and straightforward questions. But that is not always the case and underestimating questionnaire design complexity is a common and serious error when conducting surveys (SARIS and GALHOFER, 2014).

For instance, large surveys such as the Brazilian Agricultural Census (IBGE, 2017), have questionnaires that involve around a thousand quantitative measures, with intricate rules to define which questions should be part of the questionnaire, in which order those questions should be presented and what validation rules should be performed as each answer is registered. Besides the complexities hidden in the questionnaire specification, there are also a series of issues related to data collection strategy that directly affect questionnaire software requirements. For example, a survey might involve multi-mode data collection, such as the usage of self-response through the Internet and interviews conducted with the support of mobile apps, demand sampling parameters integration in data collection software or involve the usage of custom applications that contemplate requirements not related with the questionnaire[2]. Those are just a few examples of the complexities hidden in a survey questionnaire and in the software that supports it. For a full understanding of what is at stake when designing a questionnaire and a survey data collection strategy section 2.2.1 presents the survey process and section 2.2.2 sheds some light on survey IT support.

2.2.1.The survey process

The first step in understanding survey data collection complexity is to know the context in which the questionnaires are created and used: the survey process.

The United Nations Economic Commission for Europe (UNECE) built a standard business model to describe statistical production processes and its activities known as the Generic Statistical Business Process Model (GSBPM). This process was compiled from models and patterns established by statistical offices in

[2] For example, in the 2017 Brazilian Agricultural Census, before starting the questionnaire the interviewer had to execute an address confirmation with geolocation data collection. Given the cost associated with going to each and every rural property in a country, a census survey is a unique opportunity to collect all sorts of data about a country territory and its people.

an effort to establish a standard. Although very through in terms of the activities involved in running a survey, the GSBPM process does not include an activity flow and can be seen as a taxonomy that identifies the activities conducted by statistical offices (COTTON and GILLMAN, 2015). As such, in the context of this dissertation, the GSBPM fails to provide a good overview of the survey process.

Grooves et al. (2009) propose a survey process that approaches survey activities from a macro level. It provides a clear overview of the main survey activities while giving an idea of how those activities are organized in time.

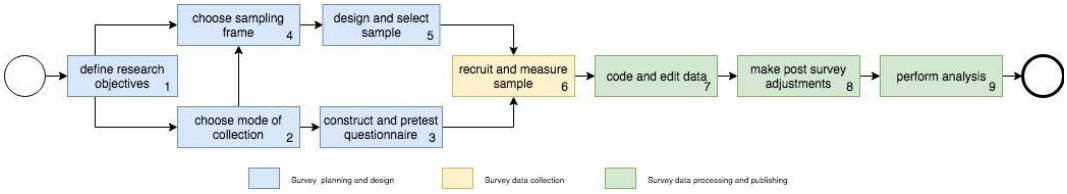

Figure 1 - Survey process adapted from Grooves et al (2009)

The proposed process starts with the research objectives definition (activity 1). With the research objectives fully understood and clearly stated, research themes to cover research objectives are selected. Next, for each theme, measures are created that express variables, conditions, criteria, causes and effects that are the focus of the survey (UNITED STATES GENERAL ACCOUNTING OFFICE, 1993). Next, activities are related to the data collection strategy definition (activities 4 and 5) and the sampling specification (activities 2 and 3). Those activities can be executed in parallel.

Defining a data collection strategy involves choosing data collection modes and making the technology to support them. These decisions are closely related to costs, questions formulation and data quality affecting all survey aspects. Next, it is necessary to translate de the concepts measured by the variables selected in activity 1 into questions, thus creating a questionnaire. Once the questionnaire is created, it has to be tested (SARIS and GALHOFER, 2014).

Sampling is closely related to data statistical analysis and it is a key step to guarantee survey quality. A sampling frame, such as a list of names and addresses of potential respondents, is needed and a procedure to select a limited number of units to describe this population must be clearly established. Choosing a sampling frame is choosing what population the survey will report on (SARIS and GALHOFER, 2014).

Once the research objectives are clearly stated, data collection is planned, questionnaire is designed/tested, and sampling is specified the survey planning is over and it is time to perform data collection (activity 6). In this stage, monitoring data collection operation is key to guaranteeing survey coverage and data quality. After the data have been collected, it goes through preliminary processing and analysis phase (activity 7). In this stage, data can be adjusted and additional data analysis can be carried out, such as coding open-ended answer questions (questions that do not have predefined choices) and performing quality checks. This is also the time for defining if it will be the necessary to generate data derived from the collected data. Based on the previous activity results, adjustments are executed to adequate the data (activity 8) and the final data analysis, with a focus on answering research questions, is performed (activity 9).

The survey process clarifies what is the questionnaire usage context. Next, it is important to understand the relationship between the survey methodology and Information Technology (IT).

2.2.2.Survey information technology support

There is no survey without some degree of IT. For simple, small scale surveys little IT support is necessary and tools used in personal computing (such as text editors and spreadsheet processors) are sufficient. As the survey goals grow in complexity and size, IT acquires relevance and IT decisions became directly related to survey feasibility. Those decisions play a relevant role in all steps of the survey process with a large influence on questionnaire design and survey data quality.

The main goal of IT in a survey process is to support the acquisition, storage, processing and analysis of the data. Figure 2 presents an example of IT architecture that support survey processes. This architecture is intended to fulfill the demands of an organization that handles concurrently the operation of multiple surveys with a high level of complexity, such as official government statistics bureaus. The architecture also presents some common issues as described next.

On the architecture diagram, each of the six ellipses represents a system: metadata system, data collection system, data collection management system, tabulation system, microdata system, publication system. The arrows connecting

systems and data storages represent data flow. Doted arrows indicate that data is not formally shared using the IT infrastructure.

In the architecture diagram three kinds of data with their respective storages are contemplated: collected data, metadata or data about the collected data, and paradata or data about the data collection process (COUPER, 1998). Collected data are core to the survey and its "*raison d'etre*". Paradata main role are to feed data collection management systems, thereby helping to detect and understand issues as the data collection operation is performed. Metadata are produced mainly during the planning phase of a survey. But, given the difficulties involved in data collection, metadata usually assume a background role and gets scattered through the survey IT architecture. This results in multiple views of what the collected data are, and it can become an inconsistency source and make data reuse harder.

It is during the survey planning phase that the metadata system realizes its main function, which is to create and store survey metadata including themes, variables, questionnaires and sampling information. Two attention points are in order when discussing survey metadata systems. First, metadata should have a single expression in the IT infrastructure and be formally integrated. If that is not the case, each of the systems in the architecture will have their own survey and questionnaire views and survey metadata gets scattered. Metadata scattering generates redundancy, rework and inconsistency among the different systems. A second issue is that whenever starting a new survey, during the planning phase, it should be easy to reuse survey metadata. But legacy systems and data pose a barrier. It is hard to keep up with technological evolution as software platforms, techniques and best practices improve and change. Still, data reuse should be always considered when evolving IT infrastructure.

After the planning phase, survey metadata is created, and data collection begins. The data collection system includes all the software in which collected data is "inputed". It is used to test the questionnaire and to perform the actual data collection operation. Both the collected data and paradata are produced by the data collection system. Besides the issues due to questionnaire complexity (this will be discussed in the next section), the main challenge posed by data collection systems is that sometimes the selected data collection strategy demands the usage of multiple technological platforms. For example, that is the case when mobile and web platforms are used concurrently to collect survey data. This scenario can easily

lead to multiple implementations based on the same questionnaire specification, which leads to multiple views of the questionnaire that should be consistent through all platforms and systems. Multiple implementations come with all the issues related to multiplatform development.

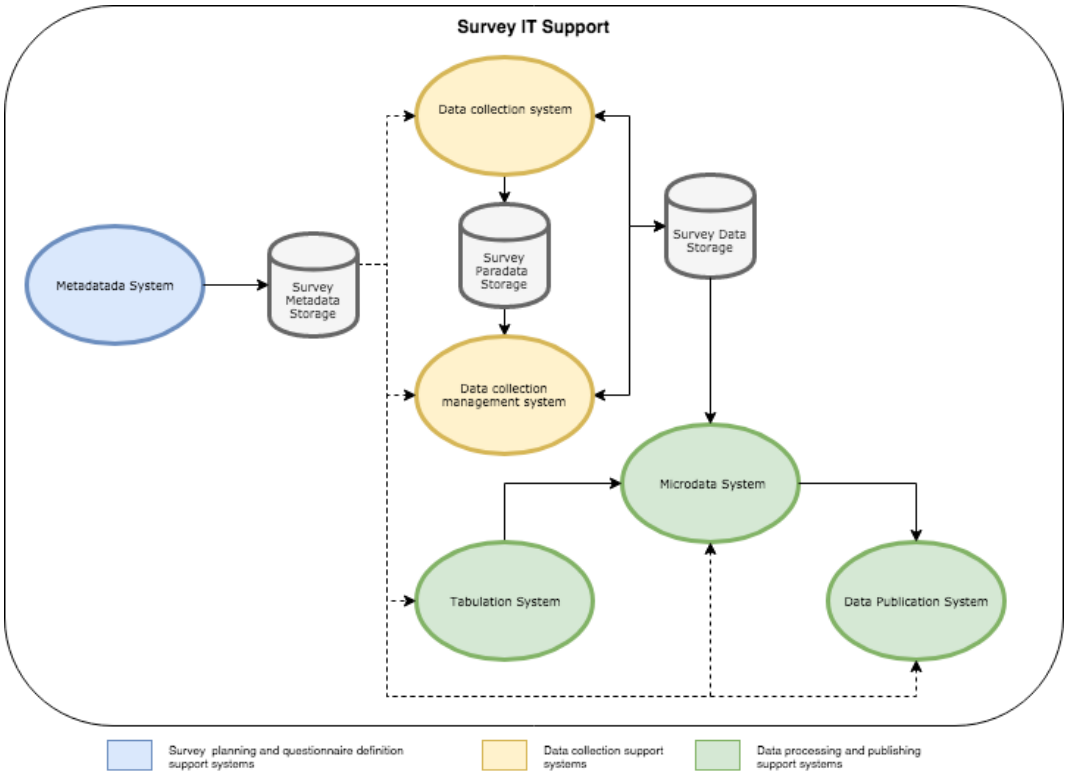

Figure 2 - IT support for the survey process

The data collection management system provides all the information and functionality to survey operation management. It is responsible for aspects such as data quality, survey coverage and fraud detection. It might include codification and imputation functionalities. Here, the main challenge is system customization since each survey will have its own monitoring demands, which derivate from the sampling frame, questionnaire specification and data collection strategy.

Finally, there are the systems related to collected data analysis and publication. Tabulation systems define tabular aggregated views for the survey data, while microdata provides access to the collected data considering security and anonymization parameters. The publication system aggregates information from microdata and tabulation to create a cohesive portray of survey results that can be shared with stakeholders.

In summary, from the architecture defined above, four challenges arise when analyzing the survey process IT infrastructure point of view. First, metadata scattering has a significant impact and generates unwanted redundancy, rework and inconsistencies. Second, systems integration and multiplatform demands can increase the IT infrastructure complexity. Third, legacy systems have a significant impact on survey metadata and data usage. Forth, since each survey is unique, there is demand for systems customization. At the center of these problems, lies survey metadata. The survey metadata is the invariant among all the systems and its central component is questionnaire definition. Thus, to further understand the challenges of a survey, one must understand the challenges of designing and using a questionnaire.

2.2.3.What is a questionnaire?

The survey definition presented in Section 2.1 highlights three survey aspects: the need of a **systematic** strategy for gathering information, the usage of **quantitative descriptors** to enable understanding this information and the fact that the information is related to the **entities** under study. In questionnaire-based surveys, the questionnaire is the protagonist. It is the questionnaire that encodes quantitative descriptors into questions about the entity attributes in which the survey is interested in. It is the questionnaire that makes data collection systematic, in the sense that each questionnaire answered will, ideally, provide a picture of a specific subject member of the population under study. It is the questionnaire that materializes the knowledge that the research aims at gathering, consolidates the research point of view for its themes as well as sampling and collected data organization (tabulations). This brief discussion brings to the forefront, the importance of questionnaires to surveys and the need for a clear definition of what is a questionnaire and a question.

The English Oxford Dictionary defines a questionnaire as “a set of printed or written questions with a choice of answers, devised for the purposes of a survey or statistical study” (QUESTIONNAIRE). Further on, a question is defined as “a sentence worded or expressed as to elicit information” (QUESTION). From these definitions, one may infer that a questionnaire is an artifact about collecting data that makes it possible to answer questions proposed by a survey. In the context of

a survey, the questionnaire is the measurement instrument (GROVES, FOWLER JR., *et al.*, 2009) and part of a larger process that aims at solving a research problem using a clearly defined methodology.

Figure 3 presents an overall picture of how questionnaires work in the context of a survey. As the survey is executed, questionnaire answers, through inference and statistical computing, are transformed in characteristics of a population. Inference is carried out in the context of a formal system that permits the description of an unobserved phenomena based on observed phenomena. The two inferential steps described in Figure 3 are a cornerstone of what a survey is, since in those steps errors arise and have to be mapped and controlled for the survey to be valid. Errors in the first step are related to the answers that people give which should accurately describe characteristics of respondents. Errors in the second step are related to the subset of persons participating in the survey which must have characteristics similar to those of a larger population (GROVES, FOWLER JR., *et al.*, 2009). This scenario makes it clear how important it is to have a well-designed questionnaire and controlled data collection, since errors in the that stage are propagated all the way up to the population characteristics with great impact on research results.

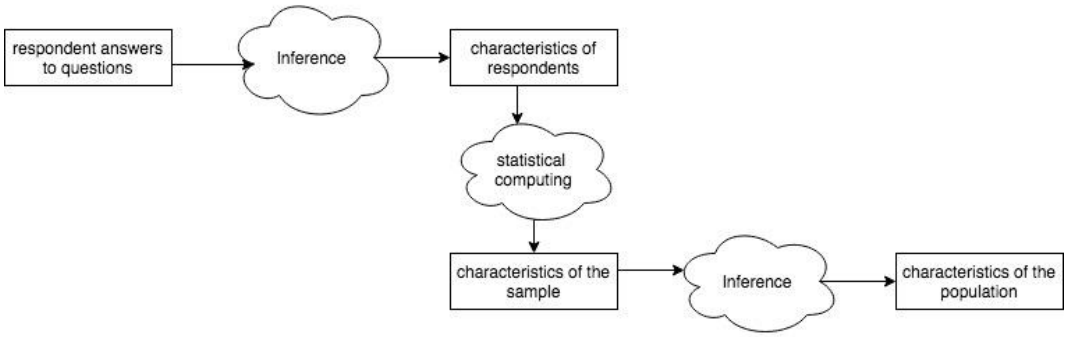

Figure 3 - Survey inference

Designing the questionnaire and questions is not simple. Nonetheless, a well-designed questionnaire is key to survey data quality. Questionnaire design starts with finding concepts that express the specified measurements. Once the concepts are defined, they must be transformed into questions. Each question aims at gathering one specific piece of data that measures the initial concept. A question

can group measurements by having one question item[3] associated with each parameter to be measured and is composed by a combination of the following: introduction, motivation, information regarding the content, information regarding definitions, instructions for the respondent, instructions for the interviewer, question items and answer choices. The process of creating questions involves five decision: 1) How to formulate a question; 2) How to organize response alternatives; 3) What will be the structure of open ended and closed questions; 4) What will be the structure of grouped (batteries) survey items; 5) What will be the order, layout and data collection method (SARIS and GALHOFER, 2014).

It is also important to have a minimum understanding of the cognitive process in answering questions and the problems in the response process that will impact survey data collection and reports. Among those problems are: failure to encode the data that the questionnaire aim at gathering into questions; misinterpretation of questions; forgetting and other memory problems while a subject provides an answer; flawed judgment or estimation strategies; problems in formatting answers; deliberate misreporting; failure to follow questionnaire instructions and proposed navigation (GROVES, FOWLER JR., *et al.*, 2009). Some of the problems pointed previously are related to the expertise in designing questions, others to the cognitive process of understanding and answering questions, and some to the operationalization of the questionnaire.

From an IT point of view, the questionnaire has a role in each of the survey process phases: first, it needs to be created during the survey planning and design phase. During the questionnaire creation, metadata, data and paradata are defined. Once the questionnaire has been created, the issue becomes how to use that questionnaire to support and control interviews during data collection phase. Considering a reality where software is mandatory, supporting the interview means transforming the defined questionnaire in a data collection application that conforms to questionnaire specification. Finally, once data are collected, the questionnaire is a reference that gives meaning to the collected data, helping to understand and produce the answers to the questions raised for the survey.

[3] Saris et al. (2014) use the term survey item to express what can be generally understood as a question. A question can have one or more requests for answers. In this work, requests for answer are called question items and are always linked to a measurement. Hence, at the end of a questionnaire-based interview, each question item will have a value associated that corresponds to the measurement.

Table 1 presents a list with survey software requirements related to questionnaires and points to the complexity in creating IT solutions to support surveys questionnaires.

Survey Questionnaire Software Development Challenges	Planning and design phase	1. Specify survey attributes
		2. Specify survey constants
		3. Specify survey tabulation
		4. Specify sampling and coverage parameters
		5. Specify themes, questions and answers considering the diversity of questioning and answering strategies
		6. Specify measurements format
		7. Specify conditional question and answer options visualization
		8. Specify data imputation according to answer and survey metadata-based rules
		9. Specify conditional questionnaire navigation (questions might be skipped depending on previous answers)
		10. Specify conditional visualization of question and answer options
		11. Specify complex data validation
		12. Specify triggers for data adjustments as questions are answered
		13. Specify interview instructions
		14. Specify creation of data derived from measurements
		15. Specify survey object of interest creation
	Data collection phase	16. Support multiplatform data collection strategies
		17. Support integration with third party software (sampling, object identification, data input software)
		18. Support question and answers customization according to context
		19. Specify paradata
		20. Support themes, questions and answers presentation customization
	Data analysis and publishing phase	21. Specify data imputation
		22. Specify coding of open text measurements

Table 1 - Questionnaire survey software requirements

An example of the requirement described in item 1 is the usage of an indicator to determine when to include a specific set of questions that measures aspects of

native population in a demographic census questionnaire. This attribute is closely related to the geographic information about the area where the interviewee lives, its value must be determined before the interview starts and be properly coded into questionnaire navigation logic to allow those questions to be asked only when necessary, helping to maintain data quality. Another example is dynamic objects of interest creation in demographic surveys (requirement item 15), such as the list of people that inhabit a household. During the interview, the person answering the questionnaire, makes a list of household inhabitants. This list is then used when questions related to education and health, for example, must be answered regarding each of the inhabitants. Those are just two examples of the challenges involved in specifying survey questionnaires.

Questionnaire complexity has a direct impact on survey IT solutions and the level of flexibility those solutions allow when creating and executing a survey. This dissertation aims at investigating the impacts that adopting a model-driven approach can have on survey support systems development and, more specifically, questionnaire specification and data collection systems development. An important step towards that is to understand how well solutions available on the market for questionnaire specification and data collection support those requirements. In the next section, those solutions are analyzed.

2.3. Software solutions for survey questionnaires

A simple search on the Web is enough to see that there are countless options when selecting software for survey data collection. On the other hand, when looking for an analysis of this class of software, the scenario is similar to what can be found in the survey methodology research field. There is little work done on comparing tools, listing requirements or establishing patterns for data collection and questionnaire design. As a result, it is clear the lack of a holistic view of survey software support.

Research work in this area is segmented with its main fields being statistical metadata systems and survey data processing, analysis and visualization. Technology evolution brought a lot of attention to data processing, analysis and visualization. This research topics, have been and still are largely researched with conferences dedicated to each of those topics.

On survey metadata there is some research work closely related to questionnaire design, but not quite the same (KARGE, 1998; VARDAKI e PAPAGEORGIOU, 2004). As already discussed in sections 2.2.2 and 2.2.3, metadata is closely related to data collection and is completed during the survey planning phase. Next comes data collection which is usually seen as a well-defined activity that is executed in the early stages of statistical surveys with benefits that do not flow into other parts of the survey process (KIM, GRUNDY and HOSKING, 2015). That interpretation on the relevance of data collection and interview software is a mistake given the importance of data collection and its impact on the survey quality. In that context and considering the list of requirements on Table 1, where does currently available survey questionnaire design and data collection software stand? Is it relevant to try a new approach or what the industry and academy now provide is enough?

Questionnaire design and data collection tools can be divided in two groups: Web based tools and frameworks. Web based tools work by allowing the user to create a questionnaire that will be distributed and answered through the Web. SurveyMonkey (SURVEYMONKEY, 2019), Zoho (ZOHO, 2019) and Qualtrics (QUALTRICS, 2019) are examples of Web based survey tools. CSPro (UNITED STATED CENSUS BUREAU, 2019), developed by the US Census Bureau, Blaise (STATISTICS NETHERLANDS, 2019), developed by Netherlands Statistics and Open Data Kit (OPEN DATA KIT, 2019) are examples of frameworks and, usually, include at least the questionnaire design tool and the data collection tool.

Although current solutions for questionnaire design and data collection have evolved, most of them do not support the level of complexity that questionnaires from large scales statistical operations require. Also, none provide a mechanism of integration with the other systems in the survey process which, as explained in Section 2.2.2, has a great impact on the survey software infrastructure. Multiplatform data collection is also an issue. Finally, all the aforementioned solutions present a tight coupling between questionnaire design and questionnaire presentation during that collection.

2.4. Model-driven Software Engineering

Model Driven Engineering (MDE) has been used by multiple engineering areas. When applied to the software development field, it receives the name of Model Driven Software Engineering (MDSE)[4]. Modeling has been part of the software development world since the early days of programming. It is a natural movement that software engineering practitioners started looking at modeling as an answer to the pressure for continuous reduction of cost and time to market, while improving software quality (BEZIVIN, 2004).

The central ideal in MDSE is to use models as first-class citizens and to transform software development in the process of creating and transforming models. As such, models constitute the main artifacts to be developed in a MDSE approach. Models assume a role that goes beyond documentation, reaching purposes such as code generation and application configuration. These new uses of models demand high-quality modeling languages capable of producing formal models that can be processed by tools (i.e., generator, interpreters, compilers, etc) (KAHLAOUI, ABRAN and LEFEBVRE, 2008).

Developers generally perceive MDSE as improving productivity, problem solving, creativity and enjoyment. But there are still barriers for its adoption such as lack of tooling to support MDSE activities and high training costs (HUTCHINSON, WHITTLE and ROUNCEFIELD, 2014).

Research regarding the benefits of MDSE adoption in large scale are limited, which makes it harder to have a clear picture of MDSE status as a software development technology. With the aim at filling this gap, a 2014 research analyzed adoption of MDSE techniques in an industrial context. The survey pointed out that the most common use of models is for problem understanding and documentation. Next comes model based code generation with testing, executable models, models for simulation and model transformations being less common. UML is the most used modelling language followed by DSLs. The study also detected that the coexistence of more than one modeling language is common. Overall, the main

[4] Nomenclature can be a challenge when discussing Model Driven Engineering in Computer Science. Some researchers use the more general term MDE, others model-driven software engineering (MDSE), others model-driven software (MDD) or model-driven software development (MDSD). In this work the option was made to use MDSE to indicate the usage of a model-driven approach when applied to software engineering in general and not just to the development phase.

motivation for model-driven engineering adoption is better communication between stakeholders, consistency among development artifacts, higher productivity, coding quality improvements and decreased time to market (WEGELER, GUTZEIT, *et al.*, 2013; MOHAGHEGHI, GILANI, *et al.*, 2013).

Although there is plenty of MDSE use cases, a lack of clarity on whether model-driven software engineering (MDSE) is a good way to develop software remains. Some companies have reported great success with it, whereas others have failed. A 2014 study on MDSE best practices points to some success factors. First, in general, companies who successfully applied MDSE in large scale did so by creating or using languages specifically developed for their domain, rather than using general-purpose modeling languages such as UML. A second factor is that MDSE tends to be most successful when driven from the ground up. MDSE efforts imposed by high level management typically struggle or fail if managers do not have the buy-in of developers first. Third, rather than following heavyweight top-down methodologies, successful MDSE practitioners use MDSE as and when it's appropriate and combine it with other methods in a very flexible way. Fourth, although important, code generation is not the key driver for adopting MDSE. Code generation is perceived as bringing benefits such as productivity but reports on productivity gains vary and are usually counter-balanced by costs with training and deployment of DSL-based software. In fact, the main perceived benefit from MDSE is that it makes it easier to define explicit architectures, especially when MDSE is a ground-up effort. The rigor that precise modeling imposes on developers actually forces them to develop explicit architecture descriptions, but in a way that does not impose a heavyweight and lengthy architecture definition process. (WHITTLE, HUTCHINGSON and ROUCEFIELD, 2014)

By 2019, MDSE is still in the process of consolidating its theories and methods. Although there is a lot of interest and many use cases, there is also a lot of space for modeling techniques and tools to improve their support of MDE-based software development. There is little consensus on modeling languages or tools and the effort made on standard general-purpose modeling languages, such as UML, has little impact on the industry. As a matter of fact, software designers either do not use UML, or use it only selectively and informally (WHITTLE, HUTCHINGSON and ROUCEFIELD, 2014). Still, the benefits are quite enticing when the initial barrier of adequate methods and tools has been transposed.

2.4.1.MDSE concepts

MDSE embraces different approaches for the creation of software systems starting from models. These approaches share common definitions and concepts that are central to the practice of MDSE. Among those concepts are systems, models, meta models, model transformations and modeling languages.

Modeling only makes sense when there is something to be modeled. It Does not really matter if this something is abstract of real. In the context of MDSE the object of modeling is a system, which can be defined as a generic concept for designating a software application, software platform or any other software artifact. Systems can also be "composed of" and be "related to" other systems (SILVA, 2015). The definition of what is a system is closely related to the definition of model.

A model is a set of statements about some system under study (SEIDEWITZ, 2003). As such, models are a simplification of a system built with an intended goal in mind. A system's model should be able to answer questions in place of the actual system (BÈZIVIN and GERBÉ, 2001). As a matter of fact, the model is a reduced rendering of the system that it represents and by removing or hiding details that are irrelevant for a given viewpoint, it lets us understand the system's essence more easily (SELIC, 2003). A good model allows predictions or inferences to be made using the created system abstraction, be it a real or language-based system (KÜHNE, 2006). To be useful, models must be a representation of a real system; communicate well; appeal to intuition regarding the system under study; be a trustworthy representation of the modeled system; allow to reason about the system enabling predictions about its behavior and its properties and be significantly cheaper to construct and analyze then the system under study (SEIDEWITZ, 2003).

The prefix meta is used whenever an operation is applied twice. For example, a discussion about discussions is a meta-discussion. As such, the expression meta model implies that modeling took place twice (KÜHNE, 2006). If you consider that in fact models are themselves systems (SILVA, 2015), creating a model can be seen as creating a system. As such, a meta model is the specification model for a class of systems where each system in the class is itself a valid model expressed in a certain modeling language (SEIDEWITZ, 2003). As a matter of fact, a meta model can be defined as the "model of a language of models" or as "models of modeling

languages" (FAVRÈ and NGUYEN, 2005). Meta models are models that define the structure of modeling languages (SILVA, 2015).

As mentioned in the previous section, a modeling language is defined by a meta model. In MDSE, a modeling language is the set of all possible models that are conformant with the modeling language abstract syntax, represented by one or more concrete syntaxes and that satisfy a given semantics. The pragmatics of a modeling language helps and guides how to use it in the most appropriate way (SILVA, 2015).

Figure 4 shows the relationship between system, model, meta model and modeling languages. First, the relationship "*ElementOf*" between model and modeling language means that a modeling language is a set of models. Second, the relationship "*Defines*" between meta model and modeling language means that a meta model is a model of a modeling language structure and that the modeling language is defined by the related meta model. From the previous two remarks, a meta model is a model of a set of models (a model of models). Finally, the "*ConformsWith*" relation between model and meta model means that the model should satisfy the rules defined at the level of its meta model (SILVA, 2015).

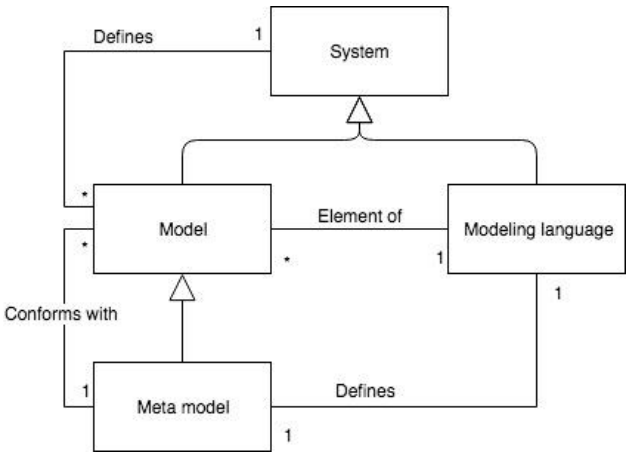

Figure 4 - System, model, meta model and modeling language relationships

For a full comprehension of what is the MDSE proposal, one final concept needs to be defined: model transformations. A model transformation, in model-driven engineering, is an automated way of modifying and creating models which can aid in ensuring that a family of models is consistent according to standards which will be defined by the software engineer. The aim of using a model

transformation is to save effort and reduce errors by automating the building and modification of models where possible (WIKIPEDIA, 2019).

Model transformations are also a way to bridge the semantic gap and move from a computer independent model (CIM), to a platform independent model (PIM). Next, from a PIM it is possible to use model transformation to move to a platform specific model (PSM), and finally, to source code as depicted in Figure 5. Transformations are the means to achieve code generation and offer a general solution for MDSE. Still it is important to keep in mind that, although code generation is a well-advertised benefit, research shows that it is not the main reason for MDSE adoption (WHITTLE, HUTCHINGSON and ROUCEFIELD, 2014).

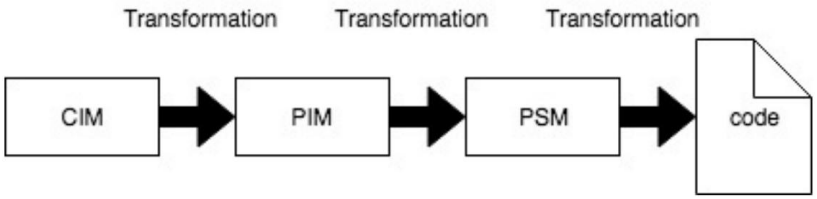

Figure 5 - Model transformations in traditional MDSE

When considering the relationship between model transformations and modeling languages it becomes evident that performing a model transformation requires a clear understanding of the abstract syntax and semantics of both the source and target modeling languages. One way to enable the creation of model transformations is through the usage of a transformation language to describe transformations with constructs for explicitly expressing, composing, and applying transformations (SENDAL and KOZACSYNSKI, 2003).

The two most common types of model transformations are Model to Text (M2T) and Model to Model (M2M) transformations. M2T is focused on generating software artifacts such as source code and other kinds of text files and the most common technique used is code generation. M2M allow translating models into another set of models, typically closer to the solution domain or that satisfy specific needs for different stakeholders. M2M approaches can be implemented with general purpose programming languages or using specialized model transformation languages such as QVT, among others.

Models, meta models, modeling languages and model transformations are traditionally explained using a meta-modeling layered architecture view of software

development. This architecture is composed of four levels. M0 is the concrete level representing the real world with situations that are unique in space and time represented by a given model. M1 is the model level. In M1 level are all models that represent real situations and that have a corresponding M2 meta model. M2 level is the meta-model level and contains any kind of meta-model. Finally, at the M3 level, one finds the meta-meta-model which is self-contained (a meta-meta-model is defined using its own primitives). Figure 6 shows this structure with the *representedBy* relationship between real systems and models and the *conformantTo* relationship characterizing the association between modeling layers and reinforcing the role of models.

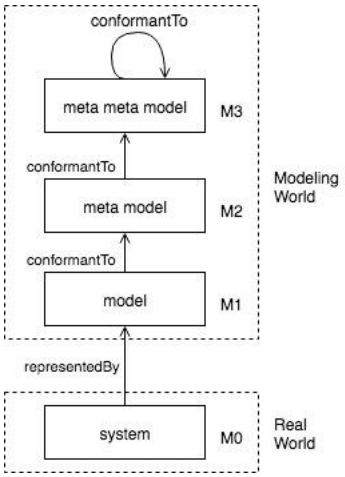

Figure 6 - MDE traditional layers

It is this layered architecture that sets the basis for MDSE approaches such as Model Driven Architecture (MDA), Microsoft Software Factories and Model Integrated Computing (BÈZIVIN, 2005). All of those solutions have in common the fact that they are general purpose model-driven software engineering solutions with different grades of tool support. For instance, MDA does not advocate specific tooling and it is an OMG standard. Although those solutions have been used and applied in the industry, their success can be seen as relative since after more than a decade, none of them have become mainstream. Research indicates that frequently success is achieved through the usage of domain-specific languages (DSLs) or domain-specific modeling languages (DSMLs) instead of using standards and general purpose modeling languages, such as UML, as meta-modeling tools (WHITTLE, HUTCHINGSON and ROUCEFIELD, 2014). As a matter of fact,

there are different approaches to MDSE and choosing which approach to use for a project can be determinant for its success.

2.4.2.MDSE approaches

Once MDSE concepts are understood and a decision is made to use it in software development, a new challenge arises. How is it done? What are the methodologies, techniques, tools? How to practice MDSE? There are a couple strategies available and quite a few tools. In fact, MDSE comes in many flavors and it is hard to make sense of all the options and technologies involved. One possibility of classification is to organize concepts, methodologies and tools in a three-level hierarchy as presented in Figure 7.

High-level approaches include the generic concepts of MDSE usually as part of standards and methodologies. At this level, there are no tools or implementation strategies. Examples are Model-driven Architecture (MDA) and Model-based Testing (MBT). Both are standards that specify concepts and structures to create Model-driven Architectures and Model-driven Testing strategies.

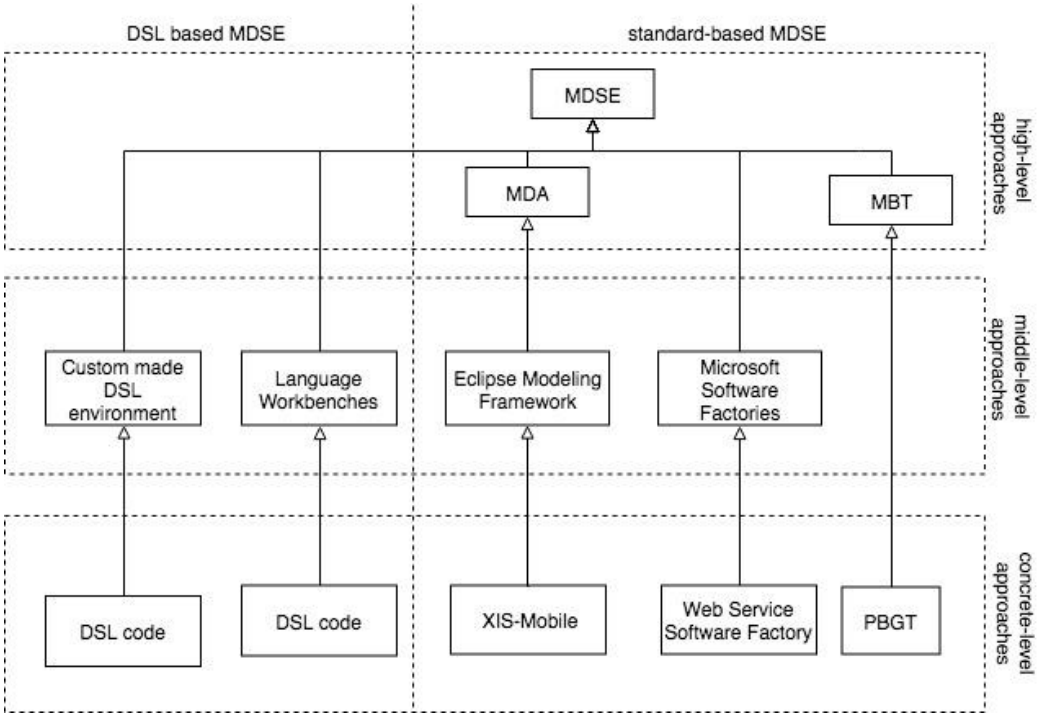

Figure 7 - MDE approaches - adapted from Silva (2015)

At the middle-level, there are tools for solving meta-model issues. Here the focus is on tools that allow users to specify meta models. Some examples are

language workbenches, the Eclipse Modeling Framework (EMF) and Microsoft Software Factories (MSF)[5].

Finally, in the concrete-level there are models created using meta models defined within the middle-layer. That is the case of DSL code created using language workbenches, XIS-Mobile (a MDD implementation that aims at increasing the productivity cross-platform mobile applications development), Microsoft Web Service Software Factory and PBGT (MBT implementation that provides generic test strategies based on user interface test patterns, with multiple configurations for testing different implementations of UI Patterns) (SILVA, 2015).

A second classification of MDSE approaches considers the impact they may have on the decision to move towards a MDSE: standard-based approaches and domain-specific approaches. While standard-based approaches leverage existing language standards (MDA/UML), tooling, and even development processes, domain-specific approaches require domain-specific languages (DSLs) and tool support to be created prior to the actual software development. The design, implementation, and testing of DSLs and their tool support require a wide spectrum of methods and techniques which bring additional complexity and challenges to the process (CZECH, MOSER and PICHLER, 2018). Standard-based approaches also tend to reinforce general modeling languages, such as UML. In comparison with domain-specific languages, that means losing freedom in expressing domain specificity. As a matter of fact, although well-known and disseminated in the industry, UML is far from being universally accepted as a MDSE approach and DSLs for narrow, well-understood domains are common. Companies who successfully applied MDE largely did so by creating or using languages specially developed for their domains, rather than using a GPML such as UML (HUTCHINSON, WHITTLE and ROUNCEFIELD, 2014).

Still, it is important to notice that is possible to apply standard-based MDSE to a specific domain. MDA/UML provides a profile mechanism that allows customizing UML while reusing its metamodel as a base language. By extending UML elements with stereotypes and their attributes, it is possible to define new concepts to better represent elements of a domain (NASCIMENTO, VIANA, *et al.*,

[5] EMF and MSF can be used to support DSL creation. In the example, they are listed in the standard-based MDSE because the concrete level solutions follow MDA e MBT.

2012). This approach can be very interesting when the DSL to be developed is not text based.

2.4.3.MDSE and survey questionnaires

MDSE is not a consensus. While some see it as a successful methodology for software development which is adopted by multiple industry areas, others claim that it failed so far, and it has limited usage. Systems generated automatically from models are rare and MDSE is far from becoming a usual method for developing systems (MUSSBACHER, AMYOT, *et al.*, 2014). Given this scenario, why think of MDSE when looking for improvements in survey questionnaire related software?

First, MDSE can be a domain-oriented technology. Surveys and survey questionnaires are a well-defined domain. As such they could benefit from known MDSE gains including higher productivity (by increased automation in the development process), increased standardization and formalism and improved communication within development teams and with external stakeholders. Besides, labor-intensive and error-prone development tasks are automated and best-known solutions can be integrated in code generators, resulting in defects reduction and software quality improvement (MOHAGHEGHI and DEHLEN, 2008). Second, MDSE allows domain experts, who specify requirements, to be directly involved in the development process because they understand the model and can work together with developers to generate code from it (BURDEN, HELDAL and WHITTLE, 2014).

To use an MDSE approach for survey questionnaires and data collection software is not a completely new idea. Kim et al. (2015) developed the Survey Design Language (SDL) and its supporting tool, SDLTool. SDL consists of a domain-specific visual language set. Each language is designed to model a specific aspect of statistical surveys providing high-level and low-level modeling facilities capable of matching experts cognitive models for statistical surveys. The SDLTool was the environment that tied together the different DSLs aspects through survey resources visual modeling, association between resources and statistical survey design elements, running modeled surveys on target population datasets, and providing visualization of the survey process. The initiative was successful but ended up discontinued since its focus was data processing, analysis and

visualization. As a matter of fact, interviewing and data collection software was perceived by the research authors as an aid with focus on the early stages of the survey process whose benefits do not flow into the survey process (KIM, GRUNDY and HOSKING, 2015). Survey methodology research shows that this perception of data collection insolated form the survey process is mistaken. Data collection, if properly modeled and implemented, can foster data quality and make statistical data production processes more efficient and effective. Since it is, indeed, part of early stages in survey methodology, it makes sense to deal with questionnaire specification and data collection first.

As seen in section 2.2.3, questionnaires can be highly complex. Still, complexity is inherent to many problems that are well solved with traditional software engineering approaches. Why then, is the adoption of MDSE for questionnaire-based surveys a good idea? First, questionnaire specification changes continuously as domain experts test and evaluate its usage. It is necessary to communicate this continuous stream of changes between software developers and domain experts. Second, integration is mandatory since data collection is only a fraction of the whole survey process. Third, having a single model for a survey questionnaire might ease the development burden when operating multi-modal data collection. Fourth, it is hard to deal with legacy collected data. Having a survey questionnaire model might provide domain experts and software developers with better tools to tame legacy data consumption issues.

One could argue that traditional software engineering practices are enough to deal with all those issues. Still, from our experience developing data collection software for censuses and large scale social, economic and demographic surveys in Brazil, that is not the case. A lot of work and time is invested in communicating with domain experts from each survey area, aligning data integration with database administrators or defining data model specifications for tabulation, imputation, microdata and data publishing systems. Documenting and tracking questionnaire requirements are a burden that takes valuable time from developers, even after data collection architecture evolved enough to allow developers to quick fix metadata directly. In fact, communication between developers and software stakeholders is a known key bottleneck in software development. A good model can describe a system behavior’s critical parts in ways a domain expert can understand and a

centralized specification can ease the communication burden (STREMBECK and ZDUN, 2009).

As mentioned, data integration is a problem as collected data needs to flow allowing the survey process to progress. Each time a domain expert changes a measurement specification, that change generates impact on database schemas that prompts database administrators to apply changes and fix inconsistent data. The ability to improve and integrate systems is always constrained by how long programmers take to figure out what the code is intended to do and how it does it. Raising the level of abstraction through the usage of models help by making it easier to understand what a questionnaire represents.

Multi-mode (which usually means multiplatform) data collection can benefit from the architectural consistency, which is a well know benefit from using MDSE approaches (HUTCHINSON, WHITTLE and ROUNCEFIELD, 2014).

Finally, having a common ground, a.k.a. questionnaire model, might help to tame legacy data. Although it is impossible to fully reconstruct questionnaires using only collected data, a lot of information can be retrieved from database schemas that might be useful to partially reconstruct old questionnaires. In software development a DSL can facilitate establishing similarities between new and past specifications enabling the reuse (ARANGO, 1994).

Those seem to be enough arguments to at least further investigate MDSE as an approach for the development of survey related software. Once the decision to investigate the usage of MDSE strategies in the development of questionnaire-based survey data collection software is made, the next question is what would be the best MDSE approach?

When trying to answer that question, it is important to remember that questionnaires are essentially specified with text. As will be seen in Chapter 3, questionnaire documentation is manly a bunch of pdfs, spreadsheets and text files. Besides, complex questionnaires present the challenge of non-linear information flows. It is known that visual representations are especially appropriate where non-linear information flows need to be expressed, be it interactions, relations or state changes. In these areas, domain-specific concrete syntax plays a key role and allow for easy recognition of important abstractions (WEGELER, GUTZEIT, *et al.*, 2013). That points us in the direction of text-based questionnaire modeling. In fact, it can be argued that textual format is more generally useful, scales better and the

necessary tools for a textual DSL take less effort to build. In the vast majority of cases, starting with textual languages is a good idea – graphical visualizations or editors can be built on top of the meta model later, when and if a real need is established. (VOELTER, BENZ, *et al.*, 2013).

A second point in defining a MDSE approach is that high-level standards and solutions play a role on guiding MDSE practice, but do not produce direct practical results which is the focus this discussion. As such, technology choices are focused on options located in the middle and concrete levels presented in Figure 7.

Third, time is a constraint in every software project. When choosing a MDSE approach productivity and level of tool support should be considered.

Fourth, it is important to delineate how much code generation will be used. Model transformation are a huge component of MDSE. But that does not mean that the only way of benefiting from it is by full-fledged generated systems. As mentioned, fully generated systems are not common and code generation, although important, it is not the first reason for adopting MDSE.

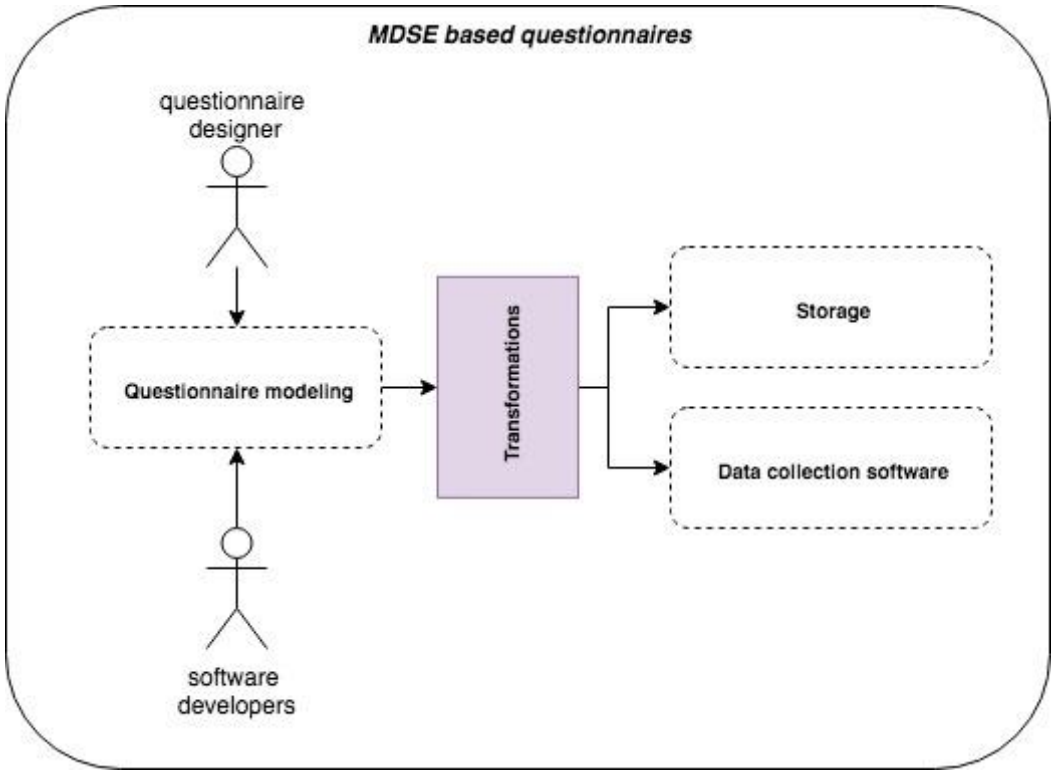

Figure 8 - Model-drive questionnaires architecture

Finally, it is important to establish properly what is the general purpose of the modeling tool. It is common sense that if one does not know what the task at hand

is, it cannot pick the best tools. Given the broad range of possibilities brought by MDSE, the choice was made to focus on the communication challenges of the questionnaire data collection software. As such DSL is mainly focused on allowing questionnaire specification and using transformations to generate partial code (domain model implementation), data collection software compatible questionnaire metadata, data collection software questionnaire configuration and data models for relational storages.

Figure 8 presents a simplified view of this model-driven questionnaire architecture.

Based on the established criteria, the choice made was to move forward with a textual DSL. Standard-based approaches with general modeling languages such as UML seem too far from questionnaire designer's communication universe, which use only text to specify and communicate questionnaire specifications. Next it is necessary to understand a little bit more about what DSLs are, how they are created, how they are used and what tools are available.

2.5. Model-driven DSL approach for survey questionnaires

There are many definitions for DSLs, but all of them are similar. Mernik *et al* (2005) define DSLs as languages tailored to a specific application domain that offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application. Deursen *et al* (2000) as a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a problem domain. Voelter *et al* (2013) define DSL as a language that is optimized for a given class of problems, called a domain. It is based on abstractions that are closely aligned with the domain for which the language is built. There are many others. In common all this DSL definitions have the fact that language universe is restricted to a specific domain. It is important to point that the first two definitions are focused on the fact that a DSL must have some level of executability which is not accurate, since there can be non-executable DSLs.

DSLs are, in general, the result of language-oriented approaches and their research field is closer to the area of programming languages. As stated in the DSL definitions, in contrast to GPLs (General Purpose Languages), DSLs are expressive

uniquely over the specific features of programs in a given problem domain, are often small, more declarative than imperative, usually textual and almost as old as GPLs. The first published papers to coin the concept of a DSL is from 1966, presenting a family of unimplemented computing languages intended to tackle differences of a given application area by a unified framework (LANDIN, 1966).

Some synonyms of DSLs are application domain languages, little or micro languages, task-specific languages, architecture description language (ADL) or specific languages. They are also closely related to scripting languages (CZECH, MOSER and PICHLER, 2018; THIBAULT, MARLET and CONSEL, 1999; NASCIMENTO, VIANA, *et al.*, 2012).

2.5.1.DSLs versus DSMLs

When starting an investigation on the field of DSLs, soon the term domain-specific modeling languages (DSMLs) will pop up. But what is a DSML? What are the differences between DSLs and DSMLS? Do they provide distinct benefits? The question about what should be used, a DSL or a DSML, is a direct consequence of knowing about the existence of DSMLs.

Over the last few decades, DSLs have proven efficient for mastering the complexities of software development projects. The natural adaptation of DSLs to the model-driven technologies has in turn established domain-specific modeling languages (DSMLs) as vital tools for enhancing design productivity (NASCIMENTO, VIANA, *et al.*, 2012). As such, DSML seem to be in fact DSLs applied in a domain-specific modeling (DSM) context (WEGELER, GUTZEIT, *et al.*, 2013).

DSM is an informal standard that prescribes an architecture and tools that aim at raising the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules form a specific problem domain. Its final goal is to generate final products in a chosen programming language or other form (KELLY and TOLVANEN, 2008).

Research shows that, with some tailoring, the same processes used to define DSLs can be used to define DSMLs, sharing the same key artifacts that are observed when an external modeling DSL is build. (STREMBECK and ZDUN, 2009).

One final remark on this subject is important. DSLs are originally a topic in the computer languages research area. As such is uncommon to see DSLs associated with term model transformations. Actually, it is also uncommon to see the term model associated to DSL code and correspondence between model and program is recent (VOELTER, BENZ, *et al.*, 2013). Hence, if models are equivalent to programs in the universe of DSLs, transformations are equivalent to code generation.

Given the above scenario, it seems irrelevant from the perspective proposed in this research to make no distinction among DSL and DSML conceptually. Also, in the bibliographic review, references using both terms were researched and are part of the theoretical basis that support the choices made.

2.5.2.DSL benefits and risks

With a clear understanding of what is a DSL, the next step is to make decisions about how to work with DSLs. To balance those decisions, it is important to be clear about the benefits and risks associated with DSL usage. Some of them are aligned with MDSE promises, others are related to the use of DSLs themselves.

The characteristics of DSLs guarantee architectures that achieve faster development of safer applications since DSLs are known to improve code quality. A DSL can also be used to parameterize a generic application. In fact, designing a DSL involves the same commonality analysis that is used in the study of a program family, i.e., determining assumptions that are true for all members of the family and variations among members. This process should be performed by both domain experts and software engineers (THIBAULT, MARLET and CONSEL, 1999). Designers must also keep in mind that DSLs can be used on different abstraction layers, ranging from technical tasks to business-level tasks. As such, DSLs should be designed to be as simple as possible while making it as powerful as needed (STREMBECK and ZDUN, 2009).

Table 2 presents the main benefits and risks associated with the decision to use a DSL as a strategy for MDSE. These benefits are closely aligned with the reasons for looking into a MDSE approach for modeling questionnaires presented in Section 2.4.3. In a broader view, the adoption of a DSL for questionnaire modeling can have its benefits reaching even further into the main challenges of

survey IT infrastructure. The usage of DSL for modeling questionnaires can standardize and centralize questionnaire metadata definition preventing metadata scattering and model transformations can be used to ease the systems integration burden. It can also bring benefits by making it easy to customize data collection software for specific survey demands. Finally, questionnaire modeling using DSL can provide a canvas for dealing with legacy data and metadata reuse.

When traditional solutions to deal with software complexity show their limitations, looking for new approaches to improve software development becomes necessary. New approaches present risks that have to be accessed and considered when adopting a new technology. From the risks listed in Table 2, the main concern when creating a DSL for questionnaire modeling are the risks associated with costs. Considering the development of data collection software for surveys with complex questionnaires, allow the survey designer control over data collection application behavior has the potential to eliminate a source of miscommunication and speedup time to market for new surveys. It can be argued that you don't need a DSL to achieve that. But any other solution will have limited potential and only mimic the behavior of DSL without the tooling infrastructure to support it. For example, many of the Web-based tools for data collection have forms for the user to configure the questionnaire. Still, none of them can handle all requirements listed on Table 1, which makes those solutions unsuitable for large scale surveys.

DSL Benefits	DSL risks
Solutions can be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs.	There are costs associated to designing, implementing and maintaining a DSL.
Programs are concise, self-documenting to a large extent, and can be reused for different purposes.	There are costs for the education for DSL users.
Enhanced productivity, reliability, maintainability, portability and reusability.	There is a limited availability of DSLs.
Domain knowledge is embodied enabling the conservation and reuse of this knowledge	It is hard to find the proper scope for a DSL.
Allow validation and optimization at the domain level.	It is hard to find the right balance between domain-specificity and general-purpose programming language constructs.

Improve testability following approaches such as MBT.	There is potential loss of efficiency when compared with hand-coded software.

Table 2 - DSLs main associated benefits and risks (DEURSEN, KLINT and VISSER, 2000; CZECH, MOSER and PICHLER, 2018; SPINELLIS, 2001; BROOKS JR, 1996)

One additional consideration on mitigating risks is that costs can be limited by limiting scope. It is always possible to start with a prototype and then move forward in case results are satisfactory.

Benefits balance out the risks when deciding to move forward with creating a DSL for questionnaire modeling. Has it already been tried? The next session discusses the use of DSLs on questionnaire modeling domain.

2.5.3.Existing DSLs for survey questionnaire modeling

DSL techniques has been applied to many domains raging from bioinformatics through robotics. In a more expressive way, web, embedded systems, low level software, control systems, parallel computing, simulation, data intensive apps, real-time systems, security, education and networks (NASCIMENTO, VIANA, *et al.*, 2012). DSLs are also not new for questionnaires, which have been the 2013 language workbench challenge (LWC) assignment. The assignment consisted of developing a DSL for questionnaires, which had to be rendered in an interactive GUI that reacted to user input, and stored answers for each question. The questionnaire definition was expected to be validated, detecting errors such as unresolved names and type errors. In addition to basic editor support, participants were expected to modularly develop a styling DSL that could be used to configure the rendering of a questionnaire. The languages created replicated basic questionnaires functionality but lacked primordial features such as the possibility to specify questionnaire navigation, complex validation rules and triggers among others (ERDWEG, STORM, *et al.*, 2015). Apart from the 2013 LWC assignment and its solutions no DSL was found for questionnaire modeling.

Some decisions patterns that point towards the usage of DSLs are closely related to questionnaire modeling. The first is DSLs being chosen due to their potential to facilitate data structure representations and transversal. The second is DSL usage to facilitate system front-end configuration and make interactions programmable (MERNIK, HEERING and SLOANE, 2005; SPINELLIS, 2001).

With the concept of DSL clearly defined, its benefits and risks understood and having found no off-the-shelf DSL for questionnaire modeling, the next step is to create one. Section 2.5.4 describes DSL development process, premises and choices that were made before starting DSL development setting the basis for the development of a questionnaire modeling language prototype.

2.5.4.DSL development process and tools

DSL development involves five phases: decision, analysis, design, implementation and deployment. At first, those steps might seem sequential, but as a matter of fact, DSL development is far from being a sequential process. For example, decision usually involves preliminary domain analysis inverting the sequential order on DSL development phases. Below is the description of what encompasses each one of those phases:

Decision: involves the process of evaluating the task at hand and choosing to solve the problem through the development of a DSL. In this process, it is important to balance benefits and risks associated with the creation and usage of the DSL.

Analysis: the problem domain is identified, domain knowledge is gathered and elaborated to generate a domain model. Domain model varies widely according to the methodology adopted in the analysis but consists of domain-specific terminology and semantics. Three patterns can be recognized when doing domain analysis for DSL construction: formal analysis, informal analysis, and extract from code (semi-automated analysis).

Design: language designer crafts DSL notation including abstract syntax, concrete syntax and static semantics. Mernik *et al* (2005) classify DSL design patterns in an orthogonal spectrum where language design strategies are positioned according to their level of formality (ranging from formal to informal) and their relationship to existing languages (ranging from language invention to language exploitation - extension, specialization or piggyback).

Implementation: this is the point at which execution environments are created for newly designed languages. For non-executable languages, infrastructure for model transformation (code generation) is established. Here again there are patterns. Table 3 presents a list of distinct approaches to DSL implementation (MERNIK, HEERING and SLOANE, 2005; STREMBECK and ZDUN, 2009; SPINELLIS, 2001).

Deployment: deployment is concerned with two aspects of DSL usage. First, delivering it to users and second DSL maintenance. Deployment is out of e scope since the DSL resulting from this research work is a prototype.

Decision criteria for developing a DSL for questionnaire modeling is based on risks and benefits as presented in Section 2.5.2. In this dissertation, the decision was made to develop a prototype for a questionnaire modeling language (SLang) and validate its usage in the context of a real data collection application (SInterviewer).

Analysis is covered in Chapter 3 and has the domain model as its output. After reviewing domain analysis methodologies and tools, the choice was made to create an informal model for SLang. In the future, that decision can be revisited in case the lack of formality becomes restrictive for DSL evolution or usage.

Design and implementation phases are interrelated since implementation provides the means to model transformations or model execution (if that is the case). In this stage a critical decision regarding the development of DSLs take place which is the choice of what tools will be used in language development. DSL design is about defining language constructs and is an activity that has a lot in common with creating programming languages in general.

The main design choices for SLang were that it would be a language invention (tool choice had an impact on this decision) done in an informal way (no mathematical models created). Three factors influenced the decision for an informal design. First the development tool choice, discussed below, brings some level of formality since the option was made to use a language workbench meta meta-model do create a questionnaire meta model. Second most questionnaire concepts are data structures that are documented and well understood. Third, questionnaires use

Boolean and arithmetic logic in questions presentation flow control that are well-know and formalizing that would be useless.

Implementation usually follows one of the approaches listed in Table 3. Each of those approaches has its strengths and weaknesses from a technical point of view. External factors, such as the technological base on which resulting DSL code will run and language purpose, are determinant in the choice of implementation strategy to be adopted for DSL implementation (VASUDEVAN, 2011). The implementation strategy also has a significant impact on the DSL development tool choices. The decision was made to implement SLang prototype using a pre-processor approach.

As the research and domain study evolved, it became clear that it doesn't make much sense to fully generate data collection applications from questionnaire models as initially planned. Also, the scope for SLang left out questionnaire presentation, limiting the possibilities for code generation.

A second important decision when planning implementation phase, is to choose the right tool. Considering time constraints, language designer expertise on target platform and level of tool support for language development and end user the decision was made to use a language workbench.

Tools for creating DSLs can be classified in two categories: DSL specification tools and language workbenches. DSL specification tools are an intuitive way of creating compilers. Once the compiler is ready, there will be no integration with other software engineering tools such as pretty printers or code assistants. JTS (Jakarta Tool Suite) is an example of this kind of tool. Language workbenches, on the other hand, support DSL creation not just in terms of parsing and code generation. A language workbench provides better editing experience for DSL users and allows DSL designers to create custom editors with functionality similar modern IDEs (NASCIMENTO, VIANA, *et al.*, 2012; CAMPAGNE, 2016).

The term language workbench was popularized by Martin Fowler (FOWLER, 2005) and refer to tools that support the efficient definition, reuse and composition of languages and their IDEs. Language workbenches make the development of new languages affordable and, therefore, support a new quality of language engineering, where sets of syntactically and semantically integrated languages can be built with comparably little effort. This can lead to multi-paradigm and language-oriented

programming environments that can address important software engineering challenges (ERDWEG, STORM, *et al.*, 2015).

Once seen as tools taking shape for the future with the promise to change the face of programming, nowadays language workbenches are complete tools that provide good support for DSL development. Still far from the predicted revolution in programming, they do introduce two powerful fusions: tool with language, and program definition with illustrative execution (FOWLER and MARTIN, 2009).

Implementation approach	Description
interpretation or compilation	DSL is execution environment is created using the classical approach to implement a new language. Standard compiler tools can be used, or tools dedicated to the implementation of DSLs.
extension of a compiler/interpreter	DSL execution environment is created by extending a GPL with domain-specific, constructs. The main advantage of this approach is that all features of the base language remain available and need not be reimplemented.
pre-processor	In this approach there is not an execution environment per si. DSL constructs are translated to constructs in an existing language by a preprocessor. The main advantage of this approach is simplicity. Its main disadvantage is that static checking and optimization are not done at the domain level. Consequently, generated code is error prone, and the user is provided with feedback on these errors at the level of the base language, or only at run-time. The four sub patterns bellow are relevant.
macro processing	Pre-processing happens through the expansion of macro definitions.
source-to-source transformation	Pre-processing happens through the transformation of DSL source code is into a base language source code.
pipeline	Pre-processing happens through a pipeline of processors that successively handle sublanguages of a DSL and translating them to the input language of the next stage.
lexical processing	Pre-processing involves only simple lexical scanning, without complicated tree-based syntax analysis.
Language specialization	Features of a base language are removed to create a DSL. A representative case arises when requirements related to the safety or security aspects of a system can be satisfied only by removing some “unsafe” aspects.

Table 3 - DSL implementation patterns

Erdwerg *et al* (2015) present a feature model and proceed the analysis of the solutions to a common problem to be solved by using a DSL and implemented in

ten distinct language workbenches. Problem considered aspects such as syntax, model execution, model validation and IDE aspects.

The top five ranked solutions included Spoofax, XText, Rascal, MetaEdit+ and MPS language workbenches. The choice was made to procced with SLang development using Meta Programming System (MPS), which is a projectional editor. A projectional editor is a user interface that makes possible to create, edit and interact with one or more ASTs avoiding the need to user tools such as parsers (CAMPAGNE, 2016).

With a clear idea of what the development process is for creating a DSL, the decision made to create the SLang questionnaire modeling language prototype and with the tool for design and implementation defined, the work proceeded to analysis phase.

3
Survey Questionnaires Domain

3.1. Introduction

Chapter 2 presented the context in which the idea of a DSL based questionnaire arises and points to the reasons why investigating the usage of a DSL based MDSE approach for survey questionnaires modeling is valuable. With the decision to develop a DSL made, the next step is to perform a domain analysis in an attempt to improve decisions about DSL design and reduce the error margin specially in what concerns what concepts should be included in the language (MERNIK, HEERING and SLOANE, 2005).

The role of domain analysis is to acquire and consolidate information about applications in the domain so that domain software infrastructure can be designed reliably. Domain analysis main goals are to establish a terminology that is sufficient for describing systems within an application domain and that is able to express all basic concepts used: entities, activities, events, and the relationships and constraints over them; to identify a set of architectures and components that can be used to assemble implementations for every specification that can be formulated in the language; to define a mapping to match specifications to relevant architectures and components unambiguously and to define those architectures and components in such a way that they can be adapted using pre-defined, minimum cost, structured mechanisms (e.g., composition, parameter instantiation, and specialization) (ARANGO, 1994).

When creating a DSL, domain analysis helps understanding clearly the components, behavior and constraints to which a domain is subjected. It helps establishing an important aspect of DSL design which is the ability to fully understand the intended language usage scope while providing means to describe the properties and the frequencies of the application in the domain, even if inaccurately (LANDIN, 1966). Before starting the domain analysis phase, it is important to have a clear view of what is a domain.

In a broad context, a domain is a sphere of activity or interest. In the context of software engineering it is most often understood as an application area, a field for which software systems are developed (PIETRO-DÍAZ, 1990).

Voelter *et al* (2013) discuss the concept of domain considering a DSL context. Defining P as the set of all conceivable programs, a domain can be specified as a subset P_D formed by all programs p_l that can be written using language l. From a DSL point of view that definition is problematic, since it can be argued whether the language l is expressive enough to cover the whole of domain D. As such, two other definitions might prove useful: one based on a bottom-up (inductive) approach and the other on a top-down (deductive) understanding of domains.

In the bottom-up approach, the domain is defined as containing all programs that are used to address a class of problems. This definition is free from any language constraint and any Turing-complete language can express those programs. But it is limited in the sense that there is no space for new or not yet solved problems and, usually, such domains do not exist outside the realm of software.

In the top-down approach the domain is defined by a body of knowledge for which software support should be provided. As such, P_D is the subset of programs in P that implement useful computations in D. The notion of domain is at the core of DSL definition. Hence, be l_D an DSL for domain D, this language should allow the creation of all programs that are members of P_D. In this definition, the domain drives the definition of the language. A second important aspect of a domain definition is all programs in the domain must be well-formed since the domain is the set of all structurally well-formed programs for an application context (JACKSON and SZTIPANOVITS, 2006). Well-formedness implies formal compliance, which requires a definition of what fits the form. As such, it is necessary to have clearly stated what are the rules that govern the domain to which models belong. That leads us to our next question. How should a domain be specified? How to describe the domain?

Domain analysis is the activity of identifying the objects and operations of a class of similar systems in a problem domain (NEIGHBORS, 1980). It involves the identification, acquisition and analysis of domain knowledge to be reused in software specification and construction (FALBO, GUIZZARDI and DUARTE, 2002). Through domain analysis information in a domain is identified, collected,

organized and represented based upon the study of existing systems and their development histories, knowledge captured from domain experts and emerging technology within a domain. As on step from Domain Engineering, domain analysis has been said to be the main process for achieving reuse success in software development (JATAIN and GOEL, 2009).

There are different approaches and processes to perform a domain analysis. In common, they have the input, which includes sources of domain knowledge, and output, which is a domain model or specification formalized with a set of artifacts. The input usually includes technical documents (requirements, system manuals, wireframes, etc.), knowledge provided by domain experts, source code, user surveys, among others. The output includes all sorts of diagrams, models and documents intended to form a cohesive description of the domain. The main goal of the domain analysis process is to find the commonalities and variabilities in software and systems that are part of the domain aiding in the reuse. The key to reusable software in domain analysis is that it focuses in the reusability of analysis and design, instead of focusing in code reuse (JATAIN and GOEL, 2009).

As mentioned above, there is not a single way of conducting domain analysis (which is highly dependent on human artistry). Researchers and practitioners propose multiple ways of doing it. From their efforts emerged methodologies, software development processes and tools designed to aid in domain analysis or in which domain analysis is a step. Among those are DARE - Domain Analysis and Reuse Environment (FRAKES, PRIETO-DIAZ and FOX, 1998), DAAS - Domain Specific Software Architectures (TRACZ, 1995), Fast - Family-Oriented Abstractions, Specification, and Translation (WEISS and LAI, 1999), FODA - Feature Oriented Domain Analysis (KANG, COHEN, *et al.*, 1990), ODE - Ontology-based Domain Engineering (FALBO, GUIZZARDI and DUARTE, 2002), and ODM - Organization Domain Modeling (SIMOS, 1995).

It is important to notice that those methodologies establish different definitions and elect distinct sets of artifacts as necessary as a result of the domain analysis process. For example, DSSA aims to reflect through the resulting model the behavior of applications in the domain under analysis. To reach that goal artifacts such as scenarios, a domain dictionary, a context (block) diagram, entity/relationship diagrams, data flow models, state transition models, and an object model are created. (TRACZ, 1995). FODA, on the other hand, defines a

domain model as a set of functions, objects, data, and relationships with a lot of effort dedicated to model features and identify which features are common and which are specific for applications under the domain. FODA domain model artifacts include features of existing software in the domain, standard vocabulary of domain experts, documentation of the entities embodied in existing software, generic software requirements via control flow, data flow, and other specification techniques (KANG, COHEN, *et al.*, 1990).

At first, the process of creating a DSL might look as a mere question of choosing an approach and tools since performing domain analysis without tool support increases the risk of failure (LISBOA, GRACIA, *et al.*, 2010). But some issues arise: how mature are domain analysis tools and how should the choice of a tool be made? Is traditional domain analysis adequate to a domain analysis aiming at creating a DSL?

Lisboa *et al.* (2010) conducted a systematic review of domain analysis tools concluding that there is no uniformity in processes supported or features offered in 15 tools analyzed. Also, gaps were perceived in the level of process support for all investigated tools. As of 2019, the scenario has evolved but has not really changed. Even though the need to clearly define the domain is recognized as a success factor for the development of a DSL (KAHLAOUI, ABRAN and LEFEBVRE, 2008), there is not a well-established process for domain modeling with a focus on designing a DSL. There is some work on systematic approaches for creating DSLs, but they do not contemplate how to perform domain analysis and prescribe generic methods such as Domain-driven Design (EVANS, 2003). There is also very little in terms of best practices on how to plan and execute domain analysis tasks when constructing a DSL (CZECH, MOSER and PICHLER, 2018).

Even though domain analysis for DSL design and implementation is still incipient, there is enough clarity on what are the basic tasks when performing it (ARANGO, 1994). Those tasks, shown in Figure 9 can be defined as (LISBOA, GRACIA, *et al.*, 2010):

- **Domain scoping**: feasibility analysis and planning the domain;
- **Data collection**: gathering information from different sources, which can vary from experts to documents;
- **Data analysis**: Description of reusable components, identifying the similarities and differences between them;

- **Classification**: classification of the information, clustering similar descriptions, abstracting relevant common features from descriptions in each cluster and vocabulary construction;
- **Evaluation of the domain model**: evaluation and correction of any defects found.

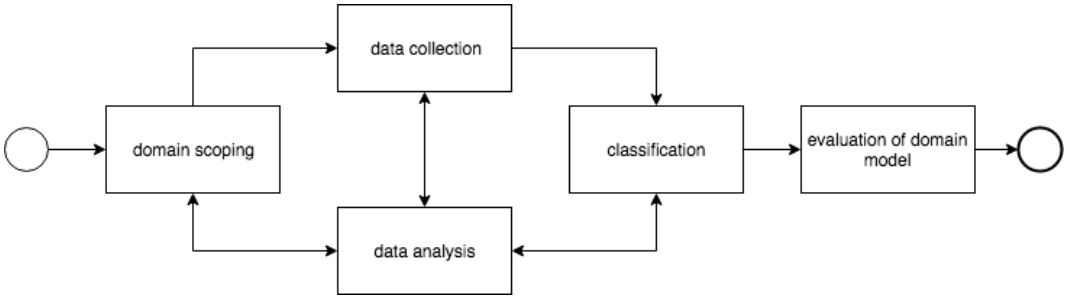

Figure 9 - Domain analysis tasks

The main goal of a domain analysis is to produce as output domain specific terminology and semantics in a more or less abstract form (MERNIK, HEERING and SLOANE, 2005). Hence, the option was made to use a combination of domain analysis methodologies and techniques, adopting the ones that brought light into survey domain aspects. To achieve that goal, the domain was analyzed and modeled based on the process described in Figure 9.

This chapter describes the results achieved and the domain model created according to the premises stated above. Section 3.2 presents the results of the domain scoping and data collection activities. A clear domain definition is presented as well as the result of data collection setting the bases for domain analysis. Section 3.3 presents the premises used in the collected data analysis as well as its main results. Section 3.4 presents the resulting domain model including its general structure, vocabulary and semantics.

3.2. Domain scope

The first task in defining the domain scope is to clearly state what is the domain all about. In the context of a DSL based survey, the term survey might be problematic since it is widely used to identify in a generic way several distinct research strategies. If using the term “statistical surveys” as a domain definition,

the adjective “statistical” helps to narrow the scope, but it is not enough since it allows the inclusion of all sorts of data collection strategies besides the usage of a questionnaire-based interview. One possibility would be to limit the domain scope to questionnaires. Again, the problem of a generic term that leaves out the concepts related to the usage of questionnaire in the context of statistical surveys would arise. The survey definition presented in Chapter 2 was too broad since it did not specify the instrument for performing data collection. Aiming at closing all the gaps, the option was made to combine both statistical survey and questionnaires in the domain scope. Thus, the scope of the domain that is focus of this research is statistical surveys in which data collection must have the following characteristics (GROVES, FOWLER JR., *et al.*, 2009):

1. Information is gathered primarily by asking people questions.
2. Information is collected either by having interviewers ask questions and record answers using forms or by having people read the questions and record their answers using forms.
3. Information is collected from (a sample of) the population to be described.

It is important to point out that, although sampling plays a big role in statistical survey methodologies, sampling is not mandatory for a survey to belong to the domain here specified.

With the domain scope clearly defined, the next step is to gather information about the domain. This activity was done in two phases. First, a research was made to locate previous domain analyses with the same or similar domain scope. Next, actual domain data was collected. Statistical surveys that are part of the domain range from simple straightforward surveys, with one theme and few questions with straightforward answers to highly complex themes, with multiple steps questions, specific question sequencing (not all questions are presented to all interviewees), multiple distinct question types, complex answer validation and imputation rules as the questionnaire based interview progresses. During data collection, the effort was in gathering information from complex surveys since their characteristics easily encompass the requirements for surveys with small, straightforward questionnaires.

3.3. Data collection

Data collection was performed in two stages. First, research was made for publications, studies and standards in which the domain defined had been analyzed and modeled. The research was conducted using keyword searching the following databases: IEEExplore, Scopus, ScienceDirect, ACM Digital Library and Springer. Search was also conducted using Google considering the possibility of domain analysis and domain models done in the industry and not reported in scientific publications. Combinations of the following keywords where used in the search: “survey”, “questionnaire”, “domain”, “domain model”, “domain analysis”, “metamodel”, “metadata” and “statistical”. In this stage, three standards and one ontology modeling the domain were found.

Next, survey and questionnaire specifications for complex surveys at IBGE were reviewed and selected. Questionnaire specifications, statistical survey data collection software requirements and documentation as well as source code where collected. Section 3.3.1 describes bibliographic research results and section 3.2.2 the collected survey specification and software development artifacts found.

3.3.1. Existing survey models

Four survey domain-related existing models were found. Three of those models are part of standards that are developed and maintained by entities related to the statistical survey community. The third one is the result of a research aiming at the development of questionnaire software for health care.

The United Nations Economic Commission for Europe (UNECE) develop and maintain a series of standards that aim at providing a common basis to survey processes that support technologies with the goal of facilitating storing, sharing and reusing statistical data and metadata. Among those standards are: Generic Statistical Information Model (GSIM), Data Documentation Initiative (also known as DDI) and Statistical Data and Metadata eXchange (SDMX) initiative.

The Generic Statistical Information Model is the first internationally endorsed reference framework of information objects, which enables generic descriptions of the definition, management and use of data and metadata throughout the statistical production process. It provides a set of standardized, consistently described

information objects, which are the inputs and outputs in the design and production of statistics. As a reference framework, GSIM helps explain significant relationships among the entities involved in statistical production and can be used to guide the development and use of consistent implementation standards or specifications (UNECE, 2019).

The Data Documentation Initiative is an international standard for describing survey metadata, including questionnaires, statistical data files, and social sciences study-level information. The DDI specification, most often expressed in XML, provides a format for content, exchange, and preservation of questionnaire and data file information supporting the description, storage, and distribution of social science data in a machine-actionable and Web-friendly manner (WIKIPEDIA, 2019; DATA DOCUMENTATION INNITIATIVE ALLIANCE, 2019).

SDMX, which stands for Statistical Data and Metadata eXchange, is an international initiative that aims at standardizing and modernizing the mechanisms and processes for the exchange of statistical data and metadata among international organizations and their member countries. It sets standards that can facilitate the exchange of statistical data and metadata using modern information technology, with an emphasis on aggregated data (WIKIPEDIA, 2019; STATISTICAL DATA AND METADATA EXCHANGE, 2019).

GSIM, DDI and SDMX include models that represent the parts of the survey and questionnaire domain with focus on a specific view. GSIM is looking at the survey process as proposed by the GSBPM. DDI is interested in providing a pattern for describing data to allow interoperability and uniformity. SDMX focus on sharing metadata and aggregated. Some overlap exists between patterns. For example, both DDI and GSIM provide elements to represent what is a questionnaire and what is a question (GSIM, 2019). Considering the goal of this research, which is to provide a DSL based approach for survey design, GISM and DDI models are relevant inputs for domain analysis. Also, the proposed language should aim at being compliant with both standards.

Besides the previously described standards, one study was found that aimed at creating a survey model. Borodin and Zavyalova (2016) describe an ontology for survey questionnaires. In the proposed ontology, a questionnaire is seen as a special method of survey using forms which supports open-ended, closed-ended (also called multivariate questions), sequencing and matching questions.

Questions are characterized by properties that describe question format (open-ended, closed-ended, sequencing or matching), their relationship to answers, responses and question sequencing (this defines questions order in the questionnaire). The ontology also defines classes that control questionnaire navigation and responses. Questionnaire navigation allows the specification of rules for skipping questions and conditional flows that customize question order. Responses allow full control of respondent feedbacks in questionnaire, question and answer level (BORODIN and ZAVYALOVA, 2016).

Qtd	Type of data	Description
6	Questionnaire specification	Actual questionnaire used by a survey including rules for sequencing questions, question and answer descriptions and some validation rules. File format varies depending on the area responsible for the questionnaire (usually done in a text file or spreadsheet).
6	Questionnaire validation rules	Specific text file containing questions validation rules.
2	Data imputation	Specific text file containing imputation rules.
6	Project source code	Source code for data collection applications. 5 data collection android-based projects and 1 data collection for the web project
1	Variable calculus rules	Text file with rules for calculating derived variables
7	Survey data dictionary	Spreadsheets describing relational database schemas used to store collected data.
7	Questionnaire metadata using json notation	Some of the analyzed statistical surveys use a json notation to store questionnaire metadata used to control the data collection application. The notation syntax and semantics varies for each survey.
4	Expert interviews	See appendix I
1	Standards	Industry specification that provides models that are included in the domain scope.

Table 4 - Data collection results

The proposed ontology works well for questionnaires with simple navigation rules, but it lacks support for: grouping questions by theme, answer validation and objects of interest creation among other requirements from complex surveys (Table 1). Also, the ontology mixes questionnaire presentation and questionnaire modeling

by specifying multiple options for closed-ended multivariate questions presentation. Still, it is a good reference and source of ideas for the survey domain model main concepts and their relationships.

3.3.2.Survey and questionnaire documents

In the second stage of the data collection effort, survey and questionnaire related documents were collected. Among the collected data were questionnaire specifications, questionnaire validation rules, data imputation rules, data collection software source code, survey data dictionaries, questionnaire metadata and interviews performed with domain specialists from different areas involved in the planning and execution of IBGE surveys. Table 4 presents a list of the collected data.

3.4. Domain model

3.4.1.Discussion

The main objective of domain data analysis is to identify key domain entities, as well as basic operations on those entities (THIBAULT, MARLET and CONSEL, 1999). It is also important to understand how entities are related and what are their frequencies (LANDIN, 1966). But, as explained in Section 3.1, there is no precise definition on what is enough to characterize a domain model (each domain analysis methodology has their own set of artifacts). As a matter of fact, domain analysis methods are a product of software reuse premises that express different views of what reuse is. As such, those methodologies are not exactly tailored for DSL creation, even if prescribed by some authors (MERNIK, HEERING and SLOANE, 2005) as the solution for DSL development analysis phase. Still domain analysis provides insights and information necessary for domain modeling.

Domain models must incorporate domain behavior and data (FOWLER, RICE, *et al.*, 2003). Its importance for creating DSLs cannot be ignored since DSLs are thin layers over models. The domain model should be designed around the purpose of the DSL and be useful without the DSL present. Fowler presents the concept of Semantic Models, which should capture all semantic behavior in a subset

of domain concepts through behaviorally rich object models that provide insights about the core behavior of a software system. Semantic models can be as simple as a data structure represented as object model, which can be manipulated in the same way as any object model, but represent only a subset of models from the domain (FOWLER and PARSONS, 2014).

Fowlers approach makes a lot of sense. Considering the goals for creating DSLs, to produce a full application model includes a series of concepts that are not always important for the DSL to be designed. For example, in the context of a questionnaire description DSL, issues pertaining question presentation and questionnaire UI are not important. Consequently, question presentation aspects won't be present in the semantic model that supports such a DSL.

With that information in mind, data classification was performed by the revision of all the collected documents and the creation of an object model aiming at fully representing questionnaire semantics.

As a criterion to decide what should be part of the model the DSL main purpose was: to provide a mean for domain experts to control questionnaire specification. Questionnaire presentation was considered out of scope as well as implementation related concepts. On the other hand, interview and answer concepts were considered part of the scope since both embrace aspects that domain experts need to control and specify.

3.4.2.Domain model structural aspects

The structural domain model is organized according to surveys three categories of information: metadata, data and paradata as presented in Section 2.1. The notation used was adapted from UML class diagrams.

Classes or concept are represented in boxes. Concepts branded with an italic font represent abstract types and interfaces. Relationship are represented by lines connecting two concepts and use the same notation for the association, aggregation, composition and generalization. Associations, aggregations and compositions present cardinalities that account for concepts frequency.

An additional type of relationship is represented with a dotted line indicating that those relationships are realized by convention without being formalized in the data collection application data structures. Such relationships bridge the boundaries

of metadata, data and paradata allowing to represent properly the existing relationships shared by those distinct classes of survey information. Domain collected data analysis shows, for example, that there is no formal link between a survey and an interview. That relationship exists only by convention. As a matter of fact, the link between a *Survey* and its associated *Interviews* is established by the fact that all interviews from a survey are stored in a specific database. In another example, the relationship between an *EntityGenerator* and an *Entity* happens only during the creation of an entity, having no other formal expression. Formal relationships are represented by solid lines. Each relationship also has an associated cardinality to indicate more clearly how the relationship frequency.

3.4.2.1.Metadata structural model

Metadata concepts define survey structure as well as consistency and integrity standards that will be applied to data during collection. Their structure is presented in Figure 10.

Survey and *Dictionary* are the root concepts that tie together the metadata concepts structure. The *Survey* concept holds general survey attributes, as well as references to survey configuration parameters, a list of constants and a list of associated metadata that allows for further survey configuration.

When creating a survey, the first important concept to be defined is *ObjectOfInterest*. Objects of interest define a class about which the survey will collect data reflecting the kind of entities the survey aims to investigate. Surveys can have one or more objects of interest, but at least one is mandatory. As an example, demographic censuses usually have at least two objects of interest: houses and people. In most cases, people are further divided into house inhabitants, deceased people and people that emigrated. Objects of interest are closely related to *EntityGenerators*.

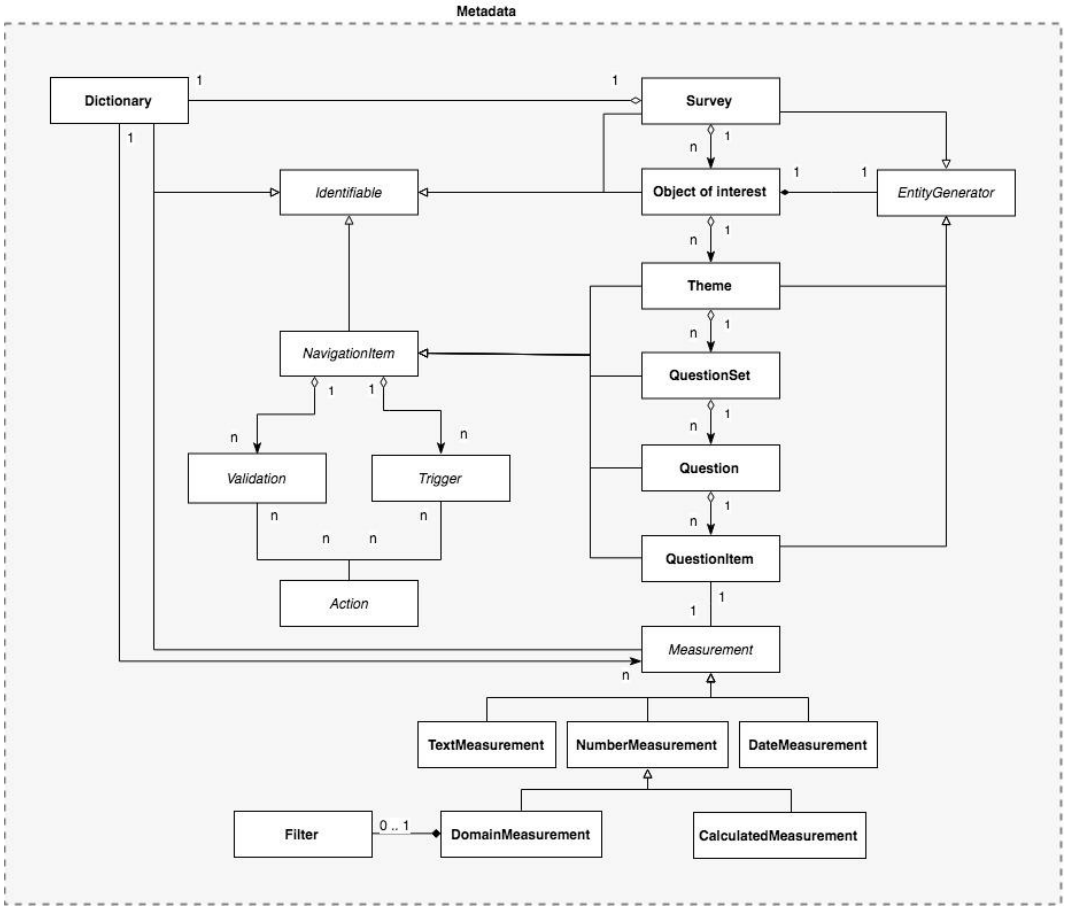

Figure 10 – Metadata domain structural model

Each object of interest holds a group of themes, each *Theme* a group of question sets, and each *QuestionSet* a group of questions. Themes and question sets are concepts responsible for the organization of questions in groups. Themes help organizing questions in terms of knowledge clusters. For example, the 2015 PeNSE survey had question organized in themes such as student socio-economical aspects, family context, eating habits, physical activity, among others. Question sets add another layer to questions organization that allows grouping questions inside a theme. For question sets, the grouping can have distinct semantics that might be defined by the questionnaire creator. The reason to create a new question set might be, for example, to create a validation rule that involves multiple measurements or hold interview instructions that are pertinent to a theme subset.

Questions are linked to a *QuestionSet* and are the core of the survey questionnaire. Each *Question* has a group of question items. Each *QuestionItem* is directly linked to a *Measurement*. At first sight, it might seem enough to link one measurement to each *Question*. But that is not true. As pointed before, a question

might comprise multiple question items (or requests for answers) (SARIS and GALHOFER, 2014). For example, consider a question about telephones in houses. Domain experts might choose to design it in two steps. In step one, the person answers if the house has a telephone line. In step two, the person answers the number (in case the answer in step one was positive).

One might wonder why the choice was made to use this concept hierarchy to organize questionnaire information. This decision was made based in the collected data, aiming at providing a structure flexible enough to support all the questionnaire samples collected. This hierarchy accounts for what a questionnaire designer might want to ask. The other side is how potential answers should be organized and all that starts with the dictionary.

Dictionary is a concept with the only function of grouping and providing a unified view of survey measurements. A *Measurement* is the survey's main goal and, in the scope of the defined domain, are taken through answers received from an individual that represents an entity under investigation. As previously noted, a *Measurement* does not have a relationship with *Question*. Measurements are linked in a one-to-one relationship with question items, where the question item is not mandatory, allowing the flexibility to create questions with multiple answers and to created derived measurements, which are not related to a question. Also, measurements must be associated to a *QuestionItem*. That is the case of measurements that are calculated from other measurements.

Measurements are part of a generalization hierarchy. The first level of this hierarchy consists of three possible data types for measurements: text (*TextMeasurement*), numbers (*NumberMeasurement*) and dates (*DataMeasurement*). Numbers can be further specialized to support domain and calculated measurements. A *DomainMeasurement* extends *NumberMeasurement* by allowing the user to associate a domain list, which is basically a map were an integer is the key and a string is the value. A *CalculatedMeasurement* has a rule that specifies what the value of the measurement should be based on questionnaire parameters and other measurement values. Filters allow controlling Domain measurements option though the association of a filter that use expressions to define if a domain item should be available to be selected as a measurement value.

Three other concepts are applied to the domain structure through generalizations, realizing important aspects of survey model: *Identifiable*, *EntityGenerator* and *NavigationItem*.

Navigation items are responsible for registering the questions presentation sequence. It is closely related to the questionnaire questions hierarchy (navigation items behavioral aspects are described in Section 3.4.4.). They hold attributes that help govern the sequence questions should be presented to the user and that allows to monitor questionnaire answer evolution (order and status attributes). Navigation items also allow to specify skip or review questions using the *go to* attribute. From a structural point of view, they also allow associating questionnaire hierarchy concepts validations and triggers, which are used to guarantee measurements consistency.

Validations allow the questionnaire designer to define when an answer is acceptable by specifying expressions. Expressions are evaluated to a Boolean value and used when defining navigation, as survey measurement values, constants and literals. Mapped Boolean operators are presented in Table 5. Three types of validation were mapped: informational validations, alert validations, error validations.

Triggers allow the execution of specific actions according to the result of a Boolean expression. One example of action is resetting previous answers. One might wonder what the difference between a trigger and a validation is. As a matter of fact, there is little difference. Trigger are simpler and dissociated from respondents' actions. As such, triggers don't have, for example, a message attribute.

An *EntityGenerator* is responsible for objects of interest creation. There are basically two object of interest creation modes. Theme-based creation and question item-based creation. Theme-based entity generators use a Theme as a basis for creating and editing objects of interest instances. Question item-based entity generators indicate an object of interest instance creation whenever a question item is answered.

An example of the first case is registering household members in a demographic census. The theme questions work as the specification of the attributes that the generated entity must have providing the basis for the creation of entities management (creating, updating and deleting) in a questionnaire.

An example of the second, is creating one object of interest item for each permanent culture an agricultural establishment grows. For example, a question might ask the respondent to select his establishment production from a list of cultures (for example, apples, avocados, grapes, oranges, peaches and pears). If the respondent selects apples and peaches, two permanent culture objects of interest should be created: one for apples and another one for peaches

Finally, each survey metadata object is an *Identifiable* which means that they can be uniquely identified in a survey metadata environment. Each Identifiable holds a code and a parent code and might be uniquely identified by the combination of its own code with its parent code.

3.4.2.2. Data and paradata structural model

Questionnaire data information follows a very simple structural model presented in Figure 11. *Interview* is the root concept and is basically a container for entities. It also keeps operational information about the questionnaire such as the last viewed item and how far the respondent has navigated the questionnaire. Each *Entity* is an instance of an object of interest and holds a set of answers.

An *Answer* holds the value from a specific measurement. In the documents and source code analyzed the relationship between answer and a measurement is in general made by convention. Hence, the option was to connect those two concepts with a dotted line. Still, it should be pointed that, for a specific interview, a measurement should be answer only one time, which accounts for the 1..1 cardinality in the relationship between measurement and answers. A second relevant point regarding this issue is that it makes no sense to talk about answer domain, filter or calculus. That is the reason why there is no parity on the Measurement and Answer specialization structure. Answer specializations are *TextAnswer*, *NumberAnswer* and *DateAnswer*. Filters are another concept that have no parity in the data structure.

Questionnaire paradata, as shown in Figure 11, holds two independent concepts: *Summary* and *Log*. The summary contains a map of aggregates that can be specified by the user. It can, for example, define a group of variables mean calculus or the number a given object of interest instances count. Aggregates are specified using rules (detailed in Section 3.4.3).

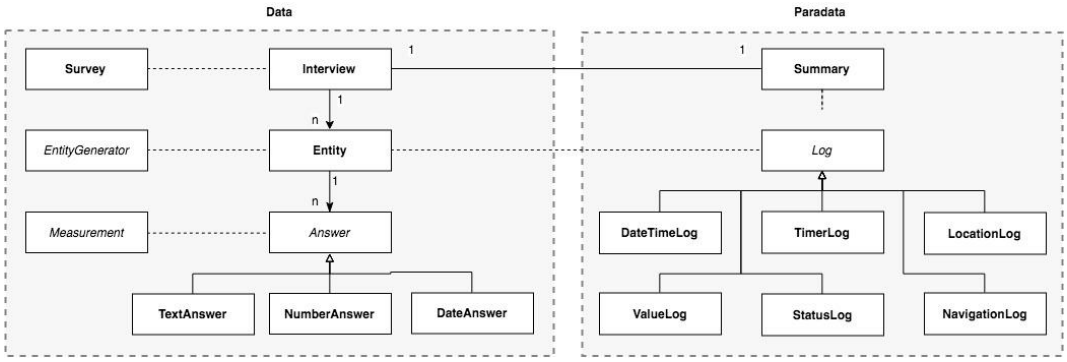

Figure 11 – Data and paradata structural model

There are six types of Logs: *DatetimeLog* holds specific information about dates, for example, the date on which an interview was initiated; *TimeLog* keeps track of time spans such as the time a respondent spends answering a question; *LocationLog* registers latitude and longitude values; *StatusLog* keeps track of status changes (section 3.4.4) as respondent is answering the questionnaire; A *ValueLog* registers changes in answers and *NavigationLog* information about the sequence themes, questions sets and questions are presented to the respondent.

The relationship between logs and interview is modeled using the weak representation of concepts association. In the analysis it was noted that due to the large number of logs compared to survey answers, logs usually receive a distinct treatment regarding storage and software design patterns. As such, it seems that they are better modeled without strong correlations with the other concepts.

3.4.3. Domain model transversal concepts

Two concepts are transversal in the sense that they are related to survey concepts but in a very independent way: *Expression* and *Rule*. Expressions evaluate to a Boolean value that is used, for example, to define validations status, *NavigationItem* visibility or if a *Question* is mandatory. Rules can be used to generate *CalulatedVariable* values or in a rule-based *Action*.

Expressions support Boolean operators, arithmetic operators, comparison operators, set operators and aggregation operators. Valid operands are literal Boolean values (true and false), strings, literal numbers (decimal or integer), measurements and survey constants. Sets are defined using parenthesis to indicate

beginning and end of a set definition and commas to separate their elements. Expressions are evaluated from left to right, using Boolean logic and arithmetic precedence rules. Parenthesis can also be used to override precedence conventions.

operators (V0001 = 5 V0002 = 1)			
type	**operator**	**example**	**result**
Boolean	and	V0001 == 5 and V0002 != 0	TRUE
	or	V0001 == 5 or V000 1 == 6	TRUE
	not	not(V0001 == 5 or V000 1 == 6)	FALSE
	empty	empty V0001	FALSE
comparison	equal	V0001 equal 6	FALSE
	not equal	V0001 not equal 6	TRUE
	greater than	V0001 greater than V0002	TRUE
	less than	V0001 less than 0	FALSE
	greater than or equal to	V0001 greater than or equal to 6	FALSE
	smaller than or equal to	V0001 smaller than or equal to 1	TRUE
arithmetic	*	V0001 * 6	30
	/	V0001 / 5	1
	-	V0001 - 5	0
	+	V0001 + 6	11
set	min	min (V0001, V0002)	1
	max	max (V0001, V0002)	5
	in	V0001 in (1,3,5)	TRUE
	out	V0001 out of (1,3,5)	FALSE
aggregation	mean	mean(V0001, V0002)	3
	sum	sum(V0001, V0002)	6

Table 5 - Operators supported by expressions

Calculus rules always evaluate to a number and use only arithmetic operators and aggregation operators. Operands might be literal numbers (decimal or integer), measurements and survey constants.

3.4.4.Domain model behavioral aspects

Each survey has a questionnaire instance with themes, questions, measurements, etc. The questionnaire data counterpart is the *Interview*. The interview is the materialization of the questionnaire as a measuring instrument. Two

important aspects of this relationship are navigation and state. As described in Section 3.4.2.1 and 3.4.2.2, triggers and validations are combined with expressions to control navigation. As the respondent navigates through the questionnaire navigation items will evolve through potential states according to respondent answers.

The domain model behavioral aspects are closely related to questionnaire navigation and to navigation items states, which are controlled by validations, triggers and expressions. Figure 12 shows the generalization relationships for *QuestionItem*, which is by inheritance a *NavigationItem*. A *QuestionItem* has two *Expression* attributes: mandatory and visibility. Visibility controls if a navigation item should be presented or not to the user according to the expression assigned to it. Mandatory attribute is used during validation cycle and be a shortcut to an actual *Validation*. The evaluation of the mandatory expression defines whether or not a *QuestionItem* should be answered.

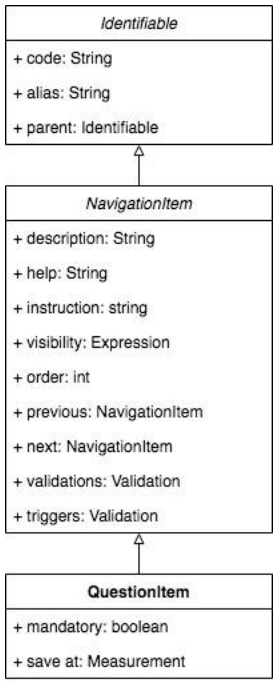

Figure 12 - Navigation item generalization

Navigation behavior is responsible for the evolution of question and questionnaire states. During an interview, usual navigation can happen vertically and horizontally from a starting point. Vertical navigation occurs whenever horizontal navigation has reached its end. Horizontal navigation happens by going

through all nested navigation items one by one (Figure 13). As such, there are two events happening: horizontal navigation events and vertical navigation events.

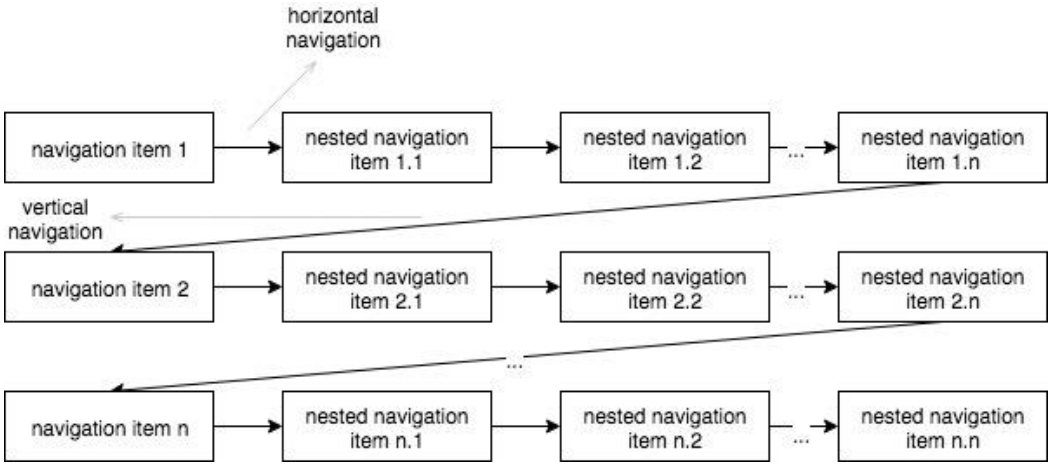

Figure 13 - Navigation sequence through navigations items hierarchy

A navigation item can have four states: NOT ANSWERED, VIEWED, SKIPPED and ANSWERED. The initial state is always NOT ANSWERED. A non-answered navigation item can either move to state VIEWED any time it is the next navigation item and its visibility evaluates to true. The non-answered navigation item can also move to state SKIPPED in case it is not visible whenever it becomes the next navigation item. A viewed item is going to stay VIEWED until it is valid, and it can be moved to ANSWERED. An answered navigation item can, during navigation, became not visible. In that case, it will move to state SKIPPED. A SKIPPED navigation item can at any time became visible and move to state VIEWED. Figure 14 presents the state diagram representing the possible state evolution for navigation items.

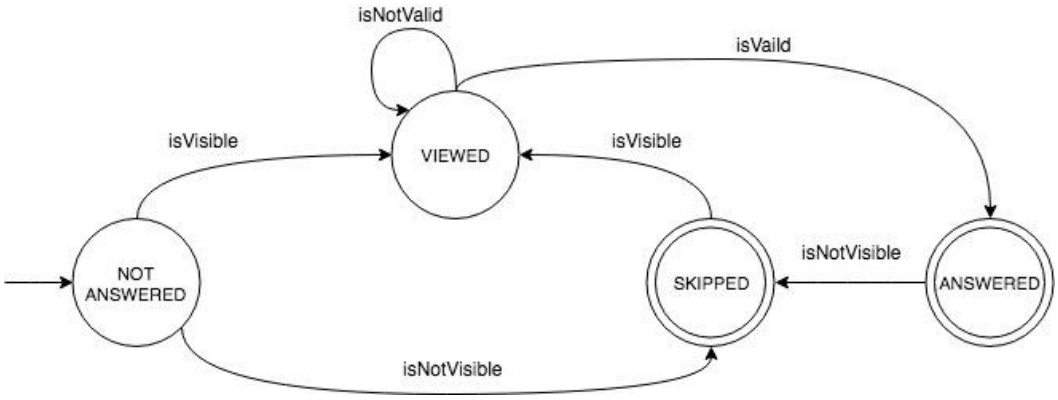

Figure 14 - Navigation items state diagram

The transitions above represented are consequence of the actions that are executed when a navigation (horizontal or vertical) happens. This sequence of actions is described in Figure 15. Whenever there is a navigation event, the first action to be executed is to run the navigation item validations. If the item is not valid. Navigation is aborted. If it is valid, triggers are run, the item is marked as ANSWERED and the next navigation item is retrieved. If the next navigation item is not visible, it is marked as SKIPPED and the next navigation item is again retrieved. When a retrieved navigation item is visible, there is a check to see if it is answered. If it is, navigation is completed. If it is not, it is first marked as VIEWED before navigation finishes.

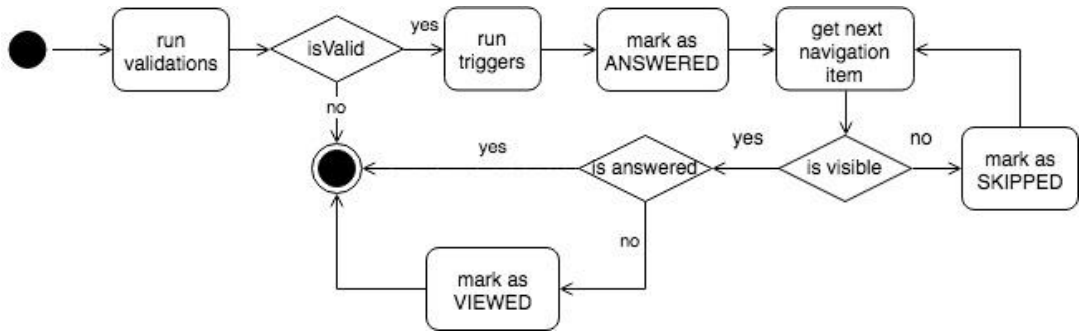

Figure 15 - Navigation activity diagram

The final aspect of questionnaire behavior is state. Questionnaire states are important for monitoring the survey (to guarantee data quality ideally you need complete questionnaires) and are registered in the *Interview* concept. Three states are possible: NOT STARTED, PENDING AND FINALIZED. Interview states are calculated based on navigation item states. An interview is marked as NOT

ANSWERED whenever all its navigation items are marked as NOT ANSWERED. It is marked as FINALIZED whenever all its navigation items are marked either as SKIPPED or ANSWERED. Finally, all other status of navigation items results in a PENDING interview.

With a good domain model, imagination can do its part to create powerful and natural data types and data structures. During implementation phase, the language designer will identify data types and data structures, naming, binding, control and syntax. Data types are a direct consequence of the application domain. Hence the importance of a good domain model. Operations follow from the data types. The notion of an operation set as being proper to a data type is very powerful. Names, binding times, naming semantics and control structures are not always mandatory for DSLs, but here the language designer has a lot of freedom. Finally, the central question when designing a language: what is a natural syntax that makes it easy to say what one means? (BROOKS JR, 1996). Chapter 4 describes the design and implementation of Slang, a prototype DSL for questionnaire specification.

4
SLang

4.1. Introduction

As presented in Chapter 2, DSLs are not new. They have been around for quite a while. There is some work on best practices (CZECH, MOSER and PICHLER, 2018) and attempts to systematize DSL design and implementation (STREMBECK and ZDUN, 2009). Also, there is guidance and documentation on what are the basic principles for implementing them. 76% of the best practices mapped by Czech *et al* (2018) where reported before 2010, indicating some level of consolidation on the subject. Still, the work on methods, processes and techniques is limited and most of the research on DSLs focuses on the creation of actual DSLs. That makes DSL implementation quite an open field where there are general guidelines, lots of experimentation and little guidance on how to choose between the available methods and tools. Most of the experimental work revolves around the creation of external[6] DSLs in domains that range from robots and control systems to Web based applications (NASCIMENTO, VIANA, *et al.*, 2012).

This scenario directly affects the ability to tackle the main challenge when creating a DSL: how does a DSL designer finds the right abstractions? Research and industry practice bring little guidance on how to map domain models to DSLs concepts and syntax. This challenge is not only a technical one, but also involves aspects of human cognition and other complex issues that are related to the difficulties of human social interaction and of natural language usage for communication (CZECH, MOSER and PICHLER, 2018; FRANK, 2011).

For those with experience in GPL development it is important to know that DSL implementation differs from the implementation of a general-purpose languages. Compilers for general purpose languages are typically structured as a lexical analyzer, a parser, a semantic analyzer, an optimizer, and a target code

6 A completely separate language, for which you write a full parser, usually using a parser generator.

generator. DSLs on the other hand have a limited scope and are usually very small. Their lexical, syntactic, and semantic simplicity often make elements required by a GPL unnecessary. In addition, the often-limited user population of a DSL does not justify high costs and a large team of developers. That reality forces DSL developers to choose cost-effective implementation strategies. Finally, when DSLs are part of the development of a larger system, schedule pressures drive language developers to implementation methods that can rapidly deliver results (SPINELLIS, 2001).

When looking for principles to support choices about DSL design and implementation, some guidance can be obtained from DSL development success factors. Among those success factors are language designer having a high level of domain expertise, a well done domain scoping, choosing the best support tools to aid in all stages of DSL development (analysis, verification, validation, code generation), use of effective meta models, good underling generator to transform DSL models into target platform code, using a high level of abstraction, development done in a domain engineering environment, having good language development expertise, use of view-point orientation to separate and organize stakeholders concerns, purpose orientation defining the particular problem in the domain the DSL aims at solving, domain expert support, usage of an effective DLS definition process (CZECH, MOSER and PICHLER, 2018).

This chapter describes the process of transforming the model created in Chapter 3 into a DSL for survey questionnaire specification. Section 4.3 details design and implementation premises already presented in Section 2.5.4. Section 4.3 DSL introduces formal definitions for some DSL design and implementation terminology. Section 4.4 describers and presents examples of SLang abstract and concrete syntaxes and Section 4.5 discusses model transformations. Finally, Section 4.6 presents SLang evaluation.

4.2. SLang design and implementation premises

Design guidelines for programming languages have been extensively discussed. Principles such as simplicity, security, fast translation, efficient object code, and readability are well established as programming languages design guidelines. For DSLs, the general principles are simplicity, uniqueness, consistency

and scalability. Still it is necessary to translate those general principles into guidelines for DSL design.

Category	Guideline
Language purpose	Identify language uses early
	Ask questions
	Make language consistent
Language realization	Decide carefully whether to use graphical or textual realization
	Compose existing languages where possible
	Reuse existing language definitions
	Reuse existing type systems
Language content	Reflect only the necessary domain concepts
	Keep it simple
	Avoid unnecessary generality
	Limit the number of language elements
	Avoid conceptual redundancy
	Avoid inefficient language elements
Concrete syntax	Adopt existing notations domain experts use
	Use descriptive notations.
	Make elements distinguishable
	Use syntactic sugar appropriately
	Permit comments
	Provide organizational structures for models
	Balance compactness and comprehensibility
	Use the same style everywhere
	Identify usage conventions
Abstract syntax	Align abstract and concrete syntax
	Prefer layout which does not affect translation from concrete to abstract syntax
	Enable modularity
	Introduce interfaces

Table 6 - DSL design guidelines

Karsai *et al* (2009) elected 26 guidelines organized into five categories: (1) **language purpose** for design guidelines applicable to the early activities of the language development process; (2) **language realization** for guidelines which discuss how to implement the language; (3) **language content** for guidelines which focus on the elements of a language; (4) **concrete syntax** for design guidelines

related to the readable (external) representation of a language; (5) **abstract syntax** concentrates on design guidelines for the internal representation of a language. While developing SLang, an effort was made to follow the guidelines on Table 6 as a mean to mitigate the risk brought by the lack of language development experience.

Decision	Criteria	Description
Concept belongs in the language?	Noteworthy level of invariant semantics	If a term can be expected to have the same meaning across all application areas a DSL is supposed to cover, it can be regarded as a candidate for being incorporated in the language.
Concept belongs in the language?	Relevance	If a concept is expected to be used on a regular base, making it part of the language would increase the languages value for most users. If it is required in rare cases only, it would increase the size of the language, hence make it more difficult to learn, while at the same time most users would not benefit from it.
Concept is a meta type or a type?	Noteworthy semantic differences between types	From a formal point of view, a meta type should allow for a range of types with clear semantic differences. If the types that can be instantiated from the potential meta type are all too similar, the effort it takes to define the types is questionable.
Concept is a meta type or a type?	Instance as type intuitive	This criterion emphasizes language ergonomics. Would it correspond to the common use of the term to regard its instances as types? Hence, would instances be regarded as types intuitively by prospective language users or would they rather be interpreted as representing instances of the respective domain?
Concept is a type or an instance?	Instance as abstraction	There are terms that emphasize abstractions by their very nature. Therefore, they resist against a clear distinction between the type and the instance level.
Concept is a type or an instance?	Invariant and unique instance identity	Sometimes, it makes sense to represent objects in a model that clearly qualify as instances. Examples include cities, countries or particular organizations. Apparently, these examples have in common that the respective instances have a unique identity that is relevant, i.e. it makes a difference, for the modelling purpose.

Table 7 - Domain model to language concepts mapping – adapted from (FRANK, 2011)

During the design phase of DSL development, the domain model must be mapped to language constructs to formally defined language abstract syntax, concrete syntax and static semantics. The focus should be on the M1 level of the layered MDSE architecture (2.4.1) and will constantly require analyzing whether a domain-level term is suited to be represented as a meta type (to be part of the language) and having a clear understanding of which concepts of your domain are potential instances (FRANK, 2011).

Table 7 presents some decision patterns that were used for guiding design choices for SLang. For example, *Interview* and *Answer* domain model concepts, were considered of low relevance for SLang users since their usage is not necessary when modeling a questionnaire. As for logs, the lack of noteworthy level of invariant semantics, the decision was made not to bring it into the language.

4.3. DSLs design and implementation terminology

When discussing DSL design and implementation it is important to set the grounds about the terminology involved. Here the main concepts related to DSL design and implementations are presented:

The *abstract syntax* is a data structure that can hold the semantically relevant information expressed by a program. It is typically a tree or a graph. It can be a synonym to metamodel and is the result of an exercise of abstraction and conceptualization followed by a synthesis of the domain knowledge done by a modeling language architect through direct interaction with the domain experts. It also includes structural or static semantics, which is mainly focused on setting binding rules among its elements (VOELTER, BENZ, *et al.*, 2013; SILVA, 2015). The *abstract syntax tree* (AST) is a data structure used to computationally represent the abstract syntax used by compilers to represent and manipulate programs. Nodes in an AST can be of different types, often arranged in one or more concept hierarchies. (CAMPAGNE, 2016)

The *concrete syntax* defines the notation with which users can express programs and, as such, defines the way users will learn and will use it, either by reading or by writing and designing the models. Notation is decisive for user experience when working with a language and it is important to look for the right balance between simplicity and expressiveness. The following concerns should be

addressed when designing a concrete syntax: writability, readability, learnability and effectiveness (VOELTER, BENZ, *et al.*, 2013; SILVA, 2015).

The *static semantics* of a language is the set of constraints and/or type system rules to which programs have to conform, in addition to being structurally correct with regards to the concrete and abstract syntax (VOELTER, BENZ, *et al.*, 2013).

The *execution semantics* refers to the meaning of a model once it is executed. It is realized using the execution engine and describes the behavior that a computer should follow when executing a model in that language. This specification can describe the relationship between the input and output of a program or can provide a step-by-step explanation on how a program will execute on a real or virtual machine (VOELTER, BENZ, *et al.*, 2013; SILVA, 2015).

The *pragmatics* of a language is related to the definition and discussion of aspects related to language use in practical contexts, namely in the definition of its types of users or roles, the activities to be conducted, and various factors that may constraint themselves (SILVA, 2015).

SLang was developed using the MPS (Section 2.5.4). To understand SLang design decisions, it is important to understand some aspects of MPS.

The first important fact is that, as a language workbench, MPS allows to construct at the same time a language and the tools to use that language. As such, during language design, meta model and model creation tool, also known as language integrated environment (IDE), are being designed. A second aspect is that MPS uses the concept of module for languages in a way that it is possible to combine existing languages while creating a new one.

MPS is a projectional editor. As such, when creating a model, the user is directly manipulating the model AST by removing or adding nodes. That means that there is no need for parsers, since the model is already represented in a tree format. Each AST node can have a parent node, child nodes, properties and references to other nodes. Nodes that do not have a parent are called root nodes which are the top elements of a language. Each node when created is associated to a concept that defines the structure of the node. As such, concepts account for the many differences among language nodes creating a “type” that connects nodes with the same structure (properties, references, etc.). Each concept can have one or more editors associated that define the way a model designer can interact with a node considering its structure. It is through the editor that the language under design has

its concrete syntax defined. By using an editor, a model designer is creating and modeling ASTs. The editor also allows to setup common integrated development environment functions such as inspector, context menus, autocompletion and context actions among others.

Language structural behavior can be constructed on top of concepts using behavior aspects. It is possible, for example, to attribute default values to properties as well as create and manipulate child nodes and references. That is done through the creation of concept constructors and concept methods. MPS offers a set of base languages, such as *sModel* and *baseLanguage*, that are very useful for AST manipulation.

Static semantics is established through constraint aspects and type system aspects. Constraint aspects help to control where concepts are allowed, valid values for properties, valid options for a reference, among others. Type system aspects define semantic aspects that cannot be modeled using concepts structure and constraints. For example, preventing nodes with the same name to exist in a specific scope cannot be done using concept structure or constraints. Constraints and type systems provide hooks that are used by MPS to provide user with static semantics functionalities, such as context assistance and error reporting, in the final language IDE generated using MPS.

MPS supports two types of model transformation: M2T and M2M. TextGen aspects are responsible for transforming an AST to text. Currently, MPS supports the generation of one text file per AST root node, by allowing the mapping of each of the root concept children to text all the way down to primitive types which have a direct conversion to text. M2M transformation are supported by the generator aspect. There are two possibilities of M2M transformations in MPS: transformations that completely preserve semantics of an input node in the output node and transformations that allow input simplification. M2M can be used in two ways: to convert language constructs from one model to another (useful for language evolution) and to convert a model between languages.

MPS has two more interesting aspects. First, MPS provides a build language that can be used to export combinations of languages into a refined IDE for the creation of models. Second, there are also tools to support language evolution through the creation of migration scripts.

4.4. SLang

SLang was constructed with focus on questionnaire specification. First, using the criteria and principles presented in Section 4.2, domain model was mapped into MPS concepts. In the process of designing the language two purposes appeared: questionnaire specification and data collection application configuration (data and paradata). It was decided to focus the work on questionnaire specification, since that is the first stone on top of which other languages can be created and combined to address issues related to other purposes. As such, SLang, at this moment, is not an executable language.

Language design was done by analyzing each domain model item and making a choice about its representation on MPS. From that, an initial conceptual structure was defined, and concrete syntax was created using MPS editors. Next, behavior and static semantics were reinforced using behavior, constraint and type system MPS aspects. This cycle was iterated a couple times until there was language complex enough to attempt language transformations. Three model transformations were implemented: SLang to SQL database schema, SLang to application specific metadata notation and SLang to java model. Finally, using MPS build language, SMaker (SLang IDE) was generated. This section presents a description of SLang with its main concepts, syntax and semantic aspects.

4.4.1. Abstract syntax

Figure 16 is a simplification of what SLang AST looks like and tries to provide an idea of how concepts are organized from an MPS point of view. In the diagrams, round rectangles represent nodes or concepts. Arrows make the link between parent and child concepts forming the AST branches. Doted arrows are used to specify non mandatory parent/child relationships. AST does not display generalization relationships which are also very important for language structure. To minimize that, concepts identified using an italic font, are abstract concepts and part of a generalization structure.

All concepts specialize the *Identifiable* abstract concept, making each one unique in the survey scope. For navigation items, *Identifiable* generates

identification automatically considering each item position in the navigation structure.

SLang root concept is *Survey*. Dictionary and Survey are related in a one-to-one composition which is slightly different from domain model design. Initially, SLang was designed to have *Dictionary* as second root concept. The option of making it a child of Survey was made after talking to domain experts, who pointed out that dictionaries are survey specific. With that in mind, the option to remove completely the dictionary concept was considered. Again, the exchange with domain experts pointed the relevance of the concept as a mean to provide a wrapper that keeps together all survey measurements. In Figure 16 simplified survey AST is possible to see how SLang reflects survey content hierarchy.

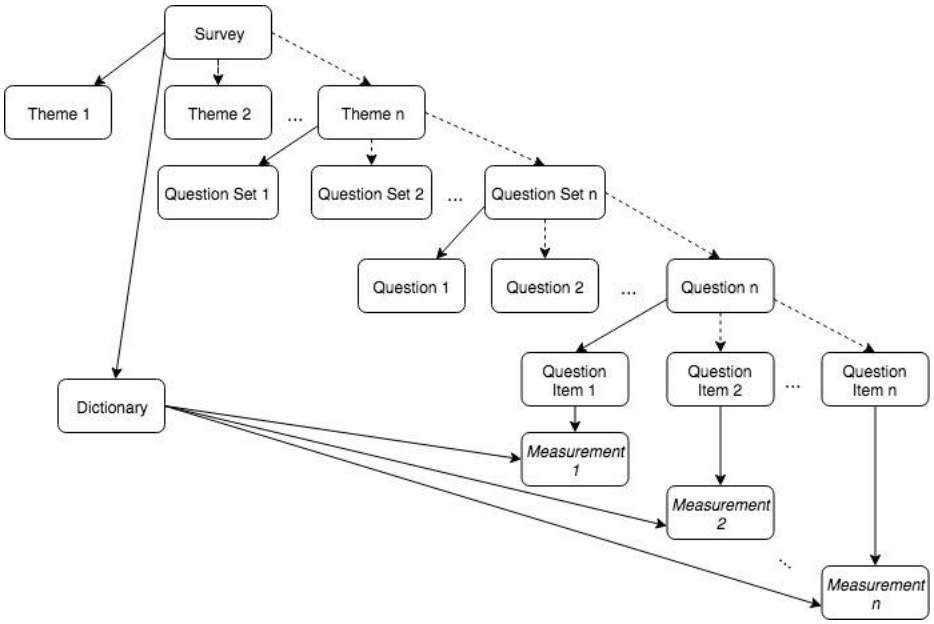

Figure 16 - Survey AST

An important aspect not shown in Figure 16 is the use of specialization. Based on MPS abstract concepts *NavigationItem* replicates the domain model design by providing the generic type that Theme, Question Set, Question and Question Item specialize in a generalization relationship equal to the one displayed on Figure 12.

Each *NavigationItem* can have multiple validations and triggers that form branches of the Survey AST. Validations have one child concept which is an *Expression*. Triggers have two child concepts: one *Expression* and one *Action*.

Actions have one child *Rule* and can have multiple child measurements. Actions are part of a generalization structure similar to domain model actions design. Those actions are used to specify changes that should happen in answers and navigation item states according to measurement values. Figure 17 represents navigation item ASTs.

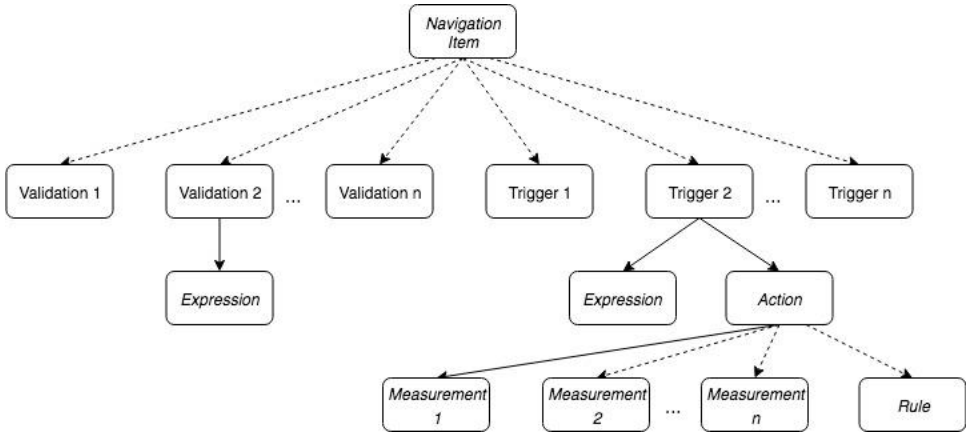

Figure 17 - Navigation Item AST

Expressions and Rules use the same constructs. As a matter of fact, Rules are an *Expression* specialization. SLang support tree primitive types: number, Boolean and text. Primitive types serve as wrappers for measurements, constants and literals which are the concepts that can be part of expressions. Operators wrap arithmetical, logic and comparison operators. A specialization of *Expression* defines the concept of *Function* that allows the implementation of set and aggregation operators (Table 5). Here implementation takes a very distinct approach from the one that should be taken in case of an executable language. Expressions, Operators and Primitives are used to check if expressions are semantically valid by match types.

As explained in Section 2.2.3, sometimes a questionnaire has more than one object of interest. In the model, this aspect is organized through the "object of interest" attribute from Themes. This attribute is mandatory and, as such, each *Theme* concept is linked to an *ObjectOfInterest* concept. It is also common the need to dynamically create objects of interest (for example, inhabitants of a household during enumerations and censuses). Two concepts can be marked as entity creators: themes and questions. Both concepts have a Boolean attribute "creator" that whenever set to true marks that concept instance as a specific object of interest

entity creator, and an object of interest attribute that indicates the type of entity to be created.

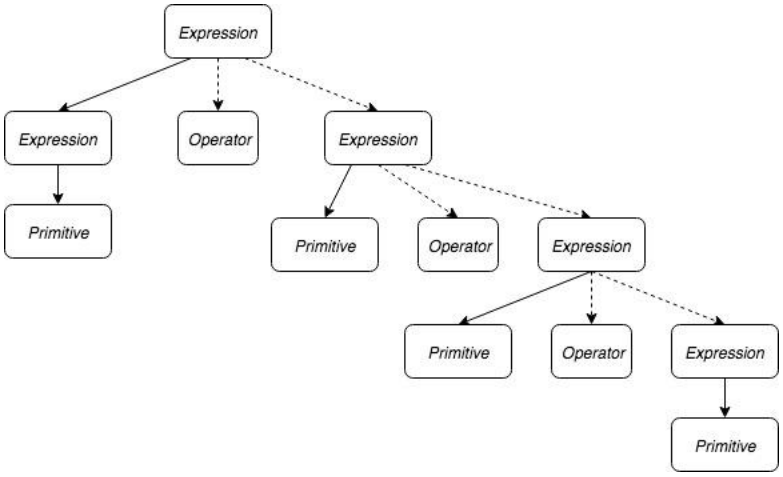

Figure 18 - Expressions AST

Objects of interest also have an important role when executing an interview, since each question context is defined by the entity (defined by an object of interest) to which the question is associated and its type. As such, SLang "object of interest" attribute in a *Theme* defines this context for purposes of questionnaire navigation. For example, it is the object of interest attribute that signals that questions under a "livestock information" theme should be replicated for each agricultural establishment that reported having livestock in a question about the establishment production (in this case, the question has the creator attribute marked as "true" and is linked). Those are the main components of SLang abstract syntax. Next, concrete syntax is presented.

4.4.2.Concrete Syntax

When starting the design of SLang concrete syntax, the first question asked was what users would expect. From conversations with domain experts, it became clear how hard that question is. The way questionnaires are expressed vary considerably depending on the user background in terms of IT tools. Some use spreadsheets, other text documents. Questionnaire creation tools or a mix of all these options, are also in order. As a result, it became very hard to get a feeling of what would be most adequate. Still, while designing concrete syntax, effort was

made to make it as close as possible to the textual writing of a questionnaire by using sensible defaults, consistent usage of style (colors, bold, italic, font), indentation and conventions. Listing 1 presents the simplest survey questionnaire that can be specified with SLang.

```
Survey: Hello World Survey (1)
  description: Hello! I'm a survey of one empty question
  version: 1 !! allows to control changes in the survey specification

  Objects of interest:
    alias: hello
    description: hello

  Theme: 1 - Hello theme
    Question Set:
      Question: Hello world!
```

Listing 1 - Hello world survey

The simplest survey must have defined properties description, a version and a code. Next, survey main concepts, Object of Interest, Theme, Question Set and Question, must be defined. For Objects of Interest alias and description are mandatory. Codes are generated automatically by SLang for all identifiables. Theme and Question have description as their mandatory attributes expressed in listing one by the strings "Hello Theme" and "Hello world!".

The principal concrete syntax definitions are main concepts, main concepts properties, identifiers, types and comments. Main concepts include: Survey, Object of Interest, Themes, Question Set, Question, Item, Trigger and Validation. Each main concept has properties that the survey designer can specify. Identifiable attributes are used in expressions to identify measurements but can also be shown for main concepts. For example, the survey designer must define a survey code. Types are used when defining survey constants or measurements. Finally, comments allow user to document SLang models and are indicated in the code with two exclamation points after any main concept or expression definition. Figure 19 presents the color scheme adopted by SLang.

Besides the mandatory description and version, surveys can have associated constants. Constants are registered in a Survey attribute with a map structure. Each

constant has an alias, a type and a value. A survey can also have multiple Objects of Interest. Each one has and alias and a description as mandatory properties. In a survey with multiple objects of interest, one of them must be marked as default by setting the default attribute to true. Listing 2 presents a Survey definition including survey constants and multiple objects of interest. For constants is possible to see its concrete syntax. The orange color indicates the constant types.

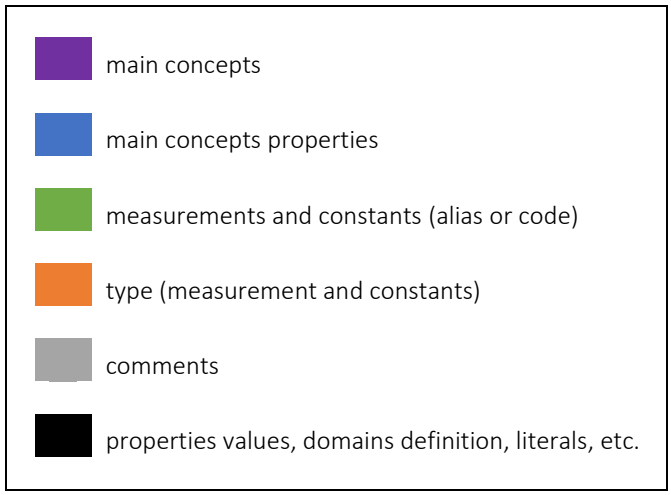

Figure 19 - Concrete syntax color conventions

Each created Survey has an associated *Dictionary*. Dictionaries are basically a list of measurements. *Measurement* attributes vary according to their type and the only non-mandatory attribute is alias. Measurements replicate the exact hierarchy from domain model (section 3.4.2) including number, text, date, domain and calculated measurements. Listing 3 presents *Dictionary* concrete syntax.

Surveys have four concepts that specialize *NavigationItem*: *Theme*, *QuestionSet*, *Question* and *QuestionItem*. For each of those concepts, attributes with sensible default values are kept hidden. The language designer changes those on demand according to expected questionnaire behavior. Survey child concepts are hierarchically organized in the following order: *Theme* is parent for *QuestionSet*, which is parent to *Question*. This hierarchy is created inside a Survey through indentation as shown in Listing 4. Attribute visibility is not shown for any of the navigation items because it is set to its default value, which is the Boolean literal "true".

```
Survey: A demographic survey (S0001)
  description: A demonstration survey
  version: 1 !! allows to control changes in the survey specification
  constants:
    min_wage: number 7.0
    reference_date: string 08/01/2019

  Objects of interest:
    alias: resident
    description: a person living in a household

    alias: household
    description: a building inhabited by people with in a known location
    default: true
```

Listing 2 – Survey coding example

Question items are child to *Question* and are questionnaire model usual place to specify measurement consistency reinforcement through the definition of validations and triggers. Listing 5 shows question items with different configurations of triggers and validations. The *Expression* typed attribute visibility is used to control which question items will be presented to the questionnaire respondent. Once visible, the mandatory attribute Expression makes sure that all these items will be answered.

```
Dictionary for: A demographic survey (D0001)
  Measurements:

  household_proprietor
  code: V0003
  type: text
  size: 100

  resident_income
  code:V0060
  type: number
  precision: 11
  scale: 2

  household_income
  code: V0060t
  type: number
  precision: 14
  scale: 2
  rule: sum(resident_income)
```

Listing 3 - SLang dictionary code

Concrete syntax contemplates two action kinds: clear and input. Trigger property expression defines when the trigger should run (trigger expression default value is true) and measurements property specifies to which measurements trigger actions should be applied.

```
Theme: Inhabitants information - T01
  object of interest: houlsehold
  instructions: Here goes instructions for a potential interviewer
  help: Here goes an explanation of this theme
    Question Set:
      Question: 1.1.1 - How many people lived in this household on {reference date}?
        Item:
          save at: number qtd_people_household

      Question: 1.1.2 - How many children with ages between zero and nine (including
newborns) lived in this household on {reference date}?

        Item:
          save at: number qtd_children_household
```

Listing 4 - SLang theme, question set and question hierarchy

Three kinds of validations are possible: INFO, ALERT and ERROR. Validations have an expression that evaluates whether measurement is consistent or not, a type indicating the level of severity of the inconsistency and a message indicating the issue causing measurement inconsistency.

The first question item in Listing 5 shows a domain measurement definition. *Measurement* domains can be edited in the associated question items or in the dictionary.

```
Question: 8.1.1 - How many sons and daughters born alive until {reference date}?
   visibility: sex == 2 && age >= 10
    mandatory: false
    Item:
       save at: domain had_children_born_alive
            1: Had Children
            2: Didn't have children
          Triggers:
            action: clear
            measurements: V0802, V0803
            expression: had_children_born_alive == 2

   Item: How many man?
         visibility: V0801 == 1
         save at: number V0802
         Triggers:
            action: input(V0802 + V0803)
           measurements: qtd_children_born_alive
         Validations:
            type: ERROR
            expression: qtd_children_born_alive > 1 && qtd_children_born_alive 30
            message: "The number of children born alive is invalid."

    Item: How many women?
         visibility: V0801 == 1
         save at: number V0803
         Triggers:
            action: input(V0802 + V0803)
            measurements: qtd_children_born_alive
```

Listing 5 - Question Items with Triggers and Validations

One last aspect of SLang concrete syntax is the specification of Objects of Interest creation. Listing 6 presents a Theme that is the creator of the *inhabitants* object of interest. This is indicated by the object of interest and creator theme attributes. As mentioned previously, all themes must have an object of interest property which specify the context of measurements. Property creator default value

is false and is only set to true for the theme or question that is responsible for entities (object of interest instances) creation.

```
Theme: Inhabitants - T02
  object of interest: inhabitant
  creator: true
  Question Set:
    Question: 2.2.1 - Inhabitants information
      Item:
        description: Inhabitant name
        save at: text name
        Validations:
          type: INFO
          expression: name in set(name)
          message: "There are two inhabitants with the same name. Do they have a last
name?”

        Item:
          description: What’s the day, month and year of your birthday?

        Item:
          description: day
          save at: number birthday_day

        Item:
          description: month
          save at: number birthday_month

        Item:
          description: year
          save at: number birthday_year

        Item:
          description: What is the relationship with the houlseholder?
          save at: domain relationship
          1: I'm the householder
          2: Partner
          3: Son
          4: Daughter
          5: Father
          Validations:
              type: ERROR
              expression: relationship == 1 && relationship in set(relationship)
              message: "A household must have just one householder"

              type: ERROR
              expression: relationship == 5 && relationship in set(relationship)
              message: "The householder already has a parent"

        Validations:
              type: ERROR
              expression: validate_date(brithday_day, birthday_month, birthday_year)
              message: "Invalid date"
```

Listing 6 - Object of interest creation specification at Theme

4.5. SLang model transformations

As any MDSE technology SLang was designed to raise the level of abstraction in data collection software development. The language by itself, might be a good exercise of modeling and language design. But the language full potential is realized by model transformations. Two model transformations where experimented with: questionnaire model to SQL schema (M2T) and questionnaire model to data collection software metadata (M2T). Experiments were also made with questionnaire model to java code (M2M), but this ended up being just a bootstrap for code and it seemed to be of little usage from a software development point of view considering the scope of the prototype. The implemented transformations are briefly described here.

The relational model is based on the concept of a relation (table) as a bidirectional structure composed of intersecting rows and columns. Columns are attributes and rows are tuples. On top of this very simple but powerful structure, a set of data manipulation constructs based on advanced mathematical concepts provide the tools to manipulate datasets (CORONEL and MORRIS, 2017).

The concepts that provide the basis to the relational model and are part of the success of relational databases is realized through the Structured Query Language (SQL). In the scope of relational databases SQL allows the creation of database and table structures, perform basic data management chores (add, delete, and modify) and perform complex queries designed to transform the raw data into useful information. It does so with minimal user effort, and its command structure and syntax are easy to learn. Finally, it is portable and conformant basic relational database management systems standards (CORONEL and MORRIS, 2017).

The questionnaire model to SQL schema transformation aims at creating the proper relational scheme to store collected data through SQL[7]. That means finding mappings between domain concepts and the relational model concepts expressed using SQL. Here the focus is on collected data structure and not on metadata or paradata structure and content.

[7] In the context of this work, the model to SQL schema transformation was crafted using SQL ANSI. It is well known that each DBMS offers flavored SQL. It is viable to tune the transformation to contemplated specific syntax depending on the target DBMS.

SLang Concept	SQL instruction
Survey	database
Object of Interest	table
Measurement	column
Validation	check
Trigger	trigger

Table 8 - SLang concepts to SQL instructions mapping

Survey and Object of Interest mappings are straight forward with the name attribute of both defining the CREATE DATABASE and a CREATE TABLE commands of SQL. Table columns are created for each measurement associated to an object of interest with appropriated type according to the measurement type attribute. Validations and triggers present the greatest challenge in this mapping because of the nature of expressions. They are respectively mapped to CHECK and CREATE TRIGGER attributes. The transformation language on MPS is straight forward although verbose. Listing 7 presents a sample from M2T transformation code.

For the second M2T transformation responsible to generate SInterviewer questionnaire specification in Json format, there was the additional challenge of deciding defaults for presentation aspects that are part of SInterviewer Json specification. These processes involved mapping SLang concepts to SInterviewer Json objects adjusting attribute names and imputing defaults where necessary. Also, SInterviewer does not accounts for the concept of a survey. Its metadata files are organized in the level of frames with one file for each theme. In that language workbench presented a limitation since M2T transformation are written to a single file. As such it was necessary to do some post processing on the resulting file to allow to break it into one file per theme.

```
base text gen component questionnaiteToSQLSchema extends <no baseTextGen> {
  operation survey(node<Survey> survey) {
    append {CREATE DATABASE } ${survey.code} {;} \n ;
    foreach objectOfInterest in survey.objectsOfInterest {
      append {CREATE TABLE } ${objectOfInterest.name} { (} \n ;
      // Primary key (object of interest code)
      append ${objectOfInterest.code} { INTEGER NOT NULL UNIQUE, } \n ;

      // If not default, add survey default object of interest key
      if (!objectOfInterest.default) {
        append ${survey.defaultObjectOfInterest.code} { INTEGER NOT NULL, } \n ;
      }
      // For each measurement add a column with the proper data type
      foreach measurement in objectOfInterest.measurements {
        append ${measurement.code} ;
        if (measurement.isInstanceOf(NumberMeasurement) ||
              measurement.isInstanceOf(DomainMeasurement) ||
              measurement.isInstanceOf(CalculatedMeasurement)) {
          append { NUMBER (} ;
          append ${String.valueOf(measurement:NumberMeasurement.precision)} ;
          append {,} ;
          append ${String.valueOf(measurement:NumberMeasurement.scale)} ;
          append {),} ;
        }
        if (measurement.isInstanceOf(TextMeasaurement)) {
          append { VARCHAR2 (} ;
          append ${String.valueOf(measurement:TextMeasaurement.length)} ;
          append {),} ;
        }
        if (measurement.isInstanceOf(DataMeasurement)) {
          append { DATE (} ;
          append ${String.valueOf(measurement:TextMeasaurement.length)} ;
          append {),} ;
        }
      }
      append \n ;
      append {PRIMARY KEY (} ${objectOfInterest.code} {),} \n ;
      if (!objectOfInterest.default) {
        append {FOREIGN KEY (} ${survey.defaultObjectOfInterest.code} {) REFERENCES }
${survey.defaultObjectOfInterest.name} ;

      }
      append {);} ;

                                                                    ...
```

Listing 7 - Sample code for M2T transformation

4.6. SLang and evaluation

DSLs are used for improving many facets of software development. Whether and to what extent this aim is achieved is an important aspect of DSL development and should be carefully addressed.

In the software engineering field, quality means good software products that meet customer expectations, constraints, and requirements while aggregating value. Despite the great variety of approaches, methods, descriptive models and tools that have been developed for software quality, a level of consensus about what means software quality has been reached by software practitioners. In MDSE field, quality continues to be a great challenge, since it is not fully defined (GIRALDO, ESPAÑA, *et al.*, 2018). Still, the number of DSLs created are increasing every day which begs the question: how DSLs should be evaluated?

There is a close relationship between quality and requirements. Requirements drive quality by establishing standards against which the conformity of something can be measured. A first issue in evaluating a DSL is related to a lack of approaches for registering DSL requirements. There is little work on how to transform domain analysis, which is well established as part of DSL development processes, into DSL requirements that can be used as a guide in an evaluation. Most of the available studies focus on general DSL requirements such as conformity, orthogonality, supportability, integrability, extensibility, longevity, simplicity, scalability and usability without providing an objective and general enough strategy for evaluating those DSL aspects (KOLOVOS, PAIGE, *et al.*, 2006).

A second relevant DSL aspect in the context of its assessment is usage context and purpose. In cases where the goal is code related developer productivity, product quality and the general usability of the tooling plays an important role. Development metrics, such as development times and code static analysis parameters, paired with developer interviews can be used to access language usability and productivity gains (KÄRNA, TOLVANEN and KELLY, 2009).

Haugen and Parastoo (2007) propose the usage of a structured questionnaire to measure DSL usability, adherence to domain and formalization[8]. Kahraman and

[8] Here formalization is related to the level of precision in language semantics. Although it might seem that formalization is always a top priority, experience shows that this is not the case. Formality is usually seen as the usage mathematical or logical terms to describe something. In the

Bilgen (2015) propose a framework for the qualitative assessment of DSLs through the usage of evaluators responsible for assessing DSL characteristics including: functional suitability, usability, reliability, maintainability, productivity, extensibility, compatibility, expressiveness, reusability and integrability. Both approaches are relevant but are highly associated with DSLs that have been deployed and for which the main goal is to generate code and fully functional systems.

Software development has well established empiricism[9] based principles. There is little hope of getting complex designs right from the first time by pure thought and ideas such as plan to throw one away, iterate and restart if necessary, early prototyping, testing with real cases, among others are part of many software development processes. Iterating designs with independent test problems is a well spread good practice. You start with a set of test problems as driving problems. Then a different set is run to see how it came out. In the long run, these examples become the driving problems of the design providing a basis for evolutionary growth (BROOKS JR, 1996).

As such SLang was evaluated by its capacity of modeling the one surveys used as reference for its domain model creation and of one survey completely new to it. The results were satisfactory with some issues on the being able to map all survey expressions. Those surveys were successfully run on SInterviewer., which is explained in the next Chapter.

universe of DSL, code generation schemes and executable models can represent formalization (HAUGEN and PARASTOO, 2007).

[9] Empiricism in the philosophy of science emphasizes evidence, especially as discovered in experiments. It is a fundamental part of the scientific method that all hypotheses and theories must be tested against observations of the natural world rather than resting solely on a priori reasoning, intuition, or revelation (WIKIPEDIA, 2019).

5
SInterviewer: A SLang use case

MDE approach claims that the use of modeling languages helps specify models in a certain level of abstraction to support the development of software applications. A “software application (or software product)” is a system composed of a nontrivial integration of software platforms, artefacts generated through model-to-text transformations, artefacts directly written by developers, and eventually models directly executable in the context of a particular software platform”. Software platforms mean an integrated set of computational elements that enable the development and execution of a class of software products (SILVA, 2015).

SInterviewer is a software application developed to allow data collection for questionnaire-based surveys on the Android platform. Its architecture is an evolution of data collection software developed for five distinct IBGE surveys including the 2017 Agricultural Census and the 2020 Demographic Census. One of the main challenges faced when developing data collection software for large scale statistical operations is the constant change in requirements. Questionnaires are continuously tested which results in changing requirements. Questionnaire specification changes management is challenging even with modern tools like distributed version control and concurrent document editing.

The next sections describe SInterviewer characteristics and main functionalities. It also discusses the improvements by the usage of SLang as reference for questionnaire specification. Section 5.1 presents SInterviewer making considerations about its main features with an emphasis on questionnaire navigation, persistence and expressions parsing followed by the description of SInterviewer architecture. Section 5.2 presents the usage of SLang as a source of questionnaire specification for SInterviewer and discusses the benefits of using SLang models to generate SInterviewer questionnaire metadata files.

5.1. SInterviewer overview

SInterviewer is data intensive mobile application built on top of the android platform. Its main purpose is the collection of questionnaire data and paradata. As mentioned in the chapter introduction, SInterviewer uses questionnaire metadata registered in Json format as the basis for delivering its features. Whenever a questionnaire is being answered, questions are shown one by one. Screen one in Figure 20 shows a simple open text question. Second screen presents questionnaire options: show questions tree (which enables to jump to any already answered question), clear current question answers, register an observation about the interview of close the questionnaire. Third screen presents the visual aspect used for validations, in this case an error alert resulting from trying to go to the next question with a mandatory field empty. In terms of question types, SInterviewer enables the usage of open text questions (numbers, text of dates), single choice questions (radio buttons or combo boxes) and multiple choices (checkboxes).

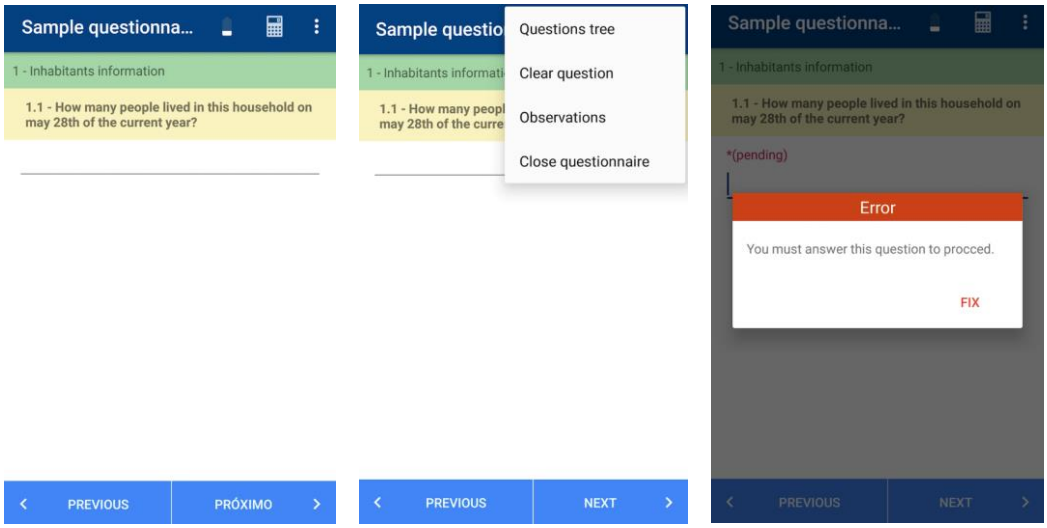

Figure 20 - SInterviewer question features

Entity editors allow dynamic creation of objects of interest. First screen on Figure 21 shows the entity manager that allows creating, editing and deleting entities of the type inhabitant. Second screen presents for inhabitant creating/editing form and third screen shows an inhabitant question.

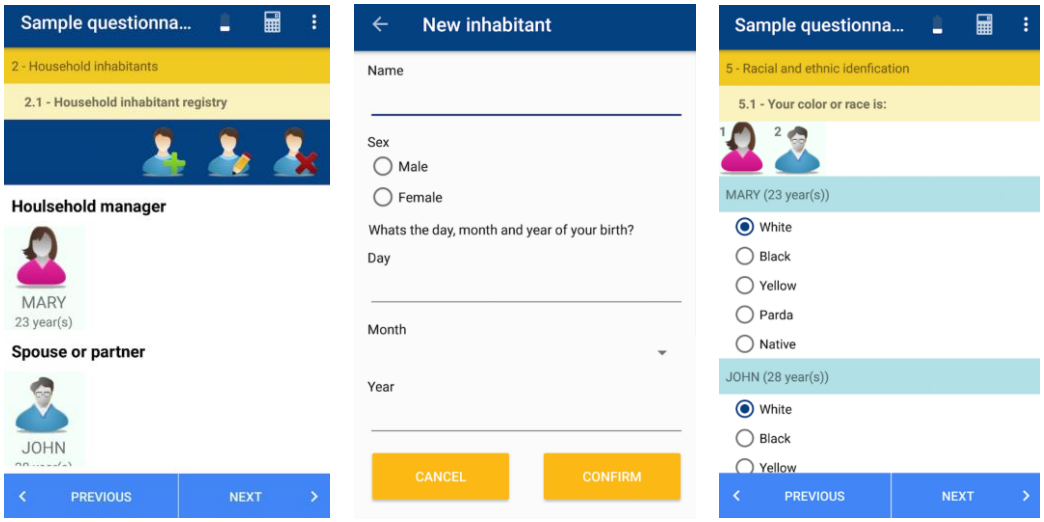

Figure 21 - Entity editor and questions

The close questionnaire function makes a final validation of the questionnaire presenting either the information that the questionnaire has been successfully answered or a list of pending questions. It is at questionnaire closing that questionnaire status is registered according to the status of its navigation items (themes, question sets, questions and question items).

SInterviewer model reflects clearly the structural domain model and is composed of three distinct groups of entities: questionnaire metadata, interview metadata and paradata related entities. Questionnaire metadata is not persisted and is read from information structured in Json files. Interview data is entered by the user as questionnaire is answered and persisted using conventions based on the measurements and dictionary specification. Paradata is partially persisted through the log entity and partially persisted using the same conventions as the interview model.

SInterviewer functionalities are based on a four layers architecture built on top of Android SDK following a flavor of the MVC architectural pattern. The top layer is responsible for presentation and governs all UI aspects. Next, the controller layer uses services layer to provide business rules and access to deserialization, expressions parsing and data persistence. Figure 22 presents a schematic of SInterviewer architecture.

Deserialization of questionnaire metadata is triggered once, when the application is started and is used through the services layer to control questionnaire navigation and answers persistence as well as general application functionality such

as closing questionnaire associated processes and interview observations registering.

SInterview supports two data models: relational data model (built on top of sqlite3) and document-based data model (built on top of Couchbase Lite). The desired data model is selected by configuration that must be made before building the application for distribution.

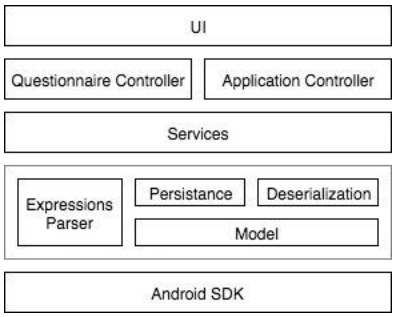

Figure 22 - SInterviewer architecture

Given the fact that questionnaires change constantly, the persistence model is not based on a direct mapping between objects with their attributes and a database model. As a matter of fact, the option was made not to code answers into object attributes and, instead, use convention for the persistence layer. For example, relational persistence layer implementation saves answers by finding a table and column given the entity name and the question item code respectively. This decision plays a big role in architectural aspects and is the reason why SInterviewer do not use the recommended Android architecture which is based on the usage of ViewModels (GOOGLE DEVELOPERS, 2019).

One of the cornerstones of a questionnaire is the possibility to use Boolean and arithmetic expressions as means to define questions navigation, answers validation and for creating derived answers based on previous collected data. SInterviewer supports the concepts of expressions through the usage of an expression's parser module. The expression's parser is a top down parser and is able to process Boolean and arithmetic expressions.

As previously explained, SInterviewer implements a model for questionnaires that is instantiated when the application is first started and questionnaire json files are deserialized. This model includes classes for some of SLang concepts such as theme, question, validation, triggers and expressions. It also includes interview and

answer entities with their associated classes. The combination of interview and questionnaire related data define questionnaire navigation. Figure 23 presents an overview of how questionnaire navigation flows after a question is answered and the user presses the next button.

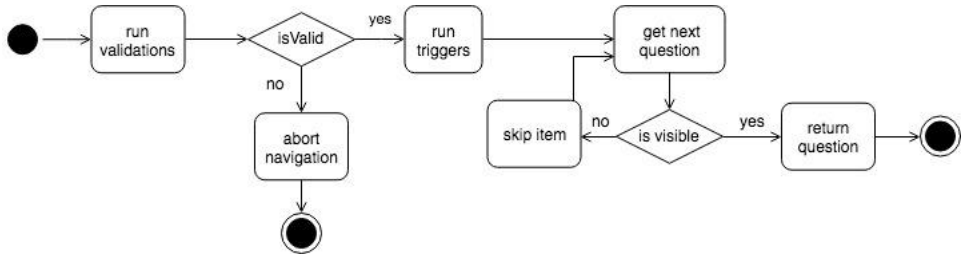

Figure 23 - SInterviewer questionnaire navigation activities

Questionnaire status control is derived from question status and is updated when questionnaire is closed. Question status is updated after validations are executed. A question validation is executed on the navigation flow and when the user closes the questionnaire.

SInterviewer supports two ways to extract data: export to CSV files and export questionnaire in a document in Json format. Those formats can be used to import questionnaire to databases or be used directly by data analysis tools.

5.2. SInterviewer and SLang

SInterviewer already has what could be seen as well-defined model, which is the Json notation used to store questionnaire metadata which works well as far as the data collection application is concerned. Then why introduce another language? What would be the role of SLang in the usage of a tool such as SInterviewer? What are the benefits and what are the disadvantages?

Before analyzing the benefits from adopting SLang as the modeling tool for questionnaires in the context of SInterviewer, it is important to clearly understand the role questionnaire Json notation plays in this scenario.

Before the current implementation used in SInterviewer, data collection software evolved slowly starting with hardcoded metadata, passing by the usage of proprietary file formats to store metadata and going all the way up to the usage of json questionnaire metadata. Each step in this evolution brought with itself

productivity and quality increases. Since Json notation has been adopted, five different surveys were implemented with a diminished time to market. Still, each of the five implemented surveys have distinct questionnaire json notations. Two factors push development teams to make changes in the notation: first, the notation mixes questionnaire model and questionnaire presentation. Second, there is not a central environment to manage the notation specification.

The first issue has not been addressed and remains an issue on SInterviewer. In what concerns the second, there has been an attempt to create a questionnaire editor. Although, the editor works well, evaluations conducted with users have shown the editor is hard to use and the way questionnaire concepts are presented is distant from the way people currently specify and document survey questionnaires and even developers avoid using the editor given how hard it is to use it to maintain the questionnaire. The next natural step on this evolution would work on improving json questionnaire notation, its management and editor usability.

The step in adopting a json notation to represent questionnaire is a clear step towards a model-driven questionnaire. Given the problems that arose, why not investigate the possibility the usage of a DSL with all it entails (IDE, debugging, syntax close to domain concepts, model transformations, etc.) as a tool to create questionnaire models that can be transformed into whatever is needed (other models, code, documentation, etc.). In that context, Language Workbenches are especially attractive, since they facilitate both defining a language and providing the tools to use the language.

Among the potential gains the adoption of a DSL offers in the specific case of SInterviewer are:

- Centralized and automated questionnaire requirements management. There are two views of questionnaire management. First there is DSL language evolution to support new concepts and relations whenever deemed necessary by language developers. That should be done guaranteeing backward compatibility. Second there is the management of questionnaire models. Questionnaire model changes can be tracked and compatibility between data collection questionnaire and specified questionnaire will be guaranteed by model transformations. As such developers no longer work with questionnaire metadata. Also, models and models components can be reused.

- Systems integration. Here model transformation plays a central role. The most basic example is the integration data collection system and microdata storage. A simple model transformation converts SLang questionnaire model to database schema represented in SQL. Another example is metadata management systems. Again, a model transformation is enough to convert questionnaire metadata to the right format survey metadata systems use.
- Automate relational schema generation for data model. That is done by using the M2T transformation that enables to generate both Json and relational schemas for data collection storages.

Those might look like small gains, but as a matter of fact each of those items brings great productivity gains when looking at large scale survey operations. Having a centralized and automated questionnaire requirements management that pretty much eliminates the need of paper documentation on questionnaire specification has the potential to eliminate most of the communication issues that generate bugs that go from wrong question texts to measurement type errors and missing question items.

SLang as a prototype is limited since it does not give its users control over questionnaire presentation aspects such as what UI components to use, visual identity and user interaction patterns. But those issues could potentially be approached by adding a questionnaire presentation model in the workflow displayed in

Figure 8. Modeling questionnaire presentation could potentially be approached by the creation of a DSL sharing the same environment used for the development of SLang.

Systems integration in the case of SInteviewer are closely related to automated relational schema generation for data model. Keeping the data under the mobile device brings little benefits for a researcher. Currently the usual approach is to have a relational database to which data collected with SInterviewer can be uploaded. Again, having the model eliminates difficulties of compatibilization between the relation schema from the data storage to data model used by SInterviewer. Being able to replicate constraints on the relational storage, adds an extra layer quality to data collection by reinforcing consistency.

The experience with SLang and SInterviewer was considered successful, but is it worth further developing SLang? Are the benefits big enough to account for costs? In the Conclusions session of this work, summarizes what was learned from this exploratory experiment, pointing to what could be the next steps considering the survey process as a role.

6
Conclusions

This dissertation presented a practical case of DSL development in the domain of questionnaire-based surveys. MDE and DSL development has evolved a lot in recent years, especially in what concerns tools for DSL creation and usage.

Although creating a DSLs it is not a novel idea, the surveys domain is still starting to realize the power of modeling languages as a tool to tame complexity (SDMX e DDI are both releasing transformation languages to facilitate metadata usage). Through the work done in this research, it was possible to form a clear view of what MDSE entails and of the potential for using DSLs as an approach for delivering MDSE in a data intensive specific domain.

The process of creating a DSL for questionnaire modeling was a challenging task given a lack of experience with computer language design. In this process, domain modeling provided a tool for domain exploration allowing a clear view of the concepts, their interrelations and frequency. Domain modeling is not new and there are many methodologies available, but none targeted at creating DSLs. The development of a specific domain modeling methodology with focus on DSL creating can, potentially, provide better focus in terms of the requirements for a creating DSL.

A second aspect with great influence on the DSL creation process was the usage of a language workbench. The language workbench provided the structure necessary to explore computer language design aspects with a clear conceptual frame in which language design and implementation had a set of premises that helped making choices. Also, the IDE (Integrated Development Environment) derived from the Language Workbench provided a tool for DSL models creation eliminating a significant amount of work in modeling tools development and diminishing the time taken to have a viable prototype.

The prototyped DSL, SLang, performed well and was evaluated by the creation of questionnaire models for some of the survey specifications collected during the analysis phase and one new survey. Although the results were acceptable,

there were shortcomings in using the current SLang concrete syntax. The questionnaire specification for the survey that was not used for domain analysis contained intricate validations rules using operations on groups of entities that could not be modeled using SLang. Also, for this questionnaire specification, validations have actions associated with user feedback upon answering a question. This last aspect falls into a gray area for SLang, since it is a transversal questionnaire specification that is closely linked with UI implementation not covered by SLang. The need of improvements was also detected as SLang should allow a questionnaire designer to specify interview instructions and have more flexibility on the usage of Expressions. All these issues can be further investigated if the option is made to start deploying SLang as a tool on an industrial scale.

The practical example in which SLang was used was quite simple, but it showed the potential for the usage of DSLs to solve practical problems, including difficult ones such as communication between domain experts and software developers. SInterviewer is still in preliminary tests and there is a long way before it is a production ready tool. In that sense, SLang also needs to evolve since, in its current state, it doesn't support questionnaire presentation specification. Still, overall, as prototypes both SLang and SInterviewer allowed to evaluate DSL usage on survey questionnaire specification domain which is a key step before advancing on the idea of deploying a DSL based tool for large scale surveys.

This research didn't cover language usability tests. Although there were informal interactions with domain experts during SLang development, that does not account for a large enough sample to validate SLang from a domain expert point of view. That kind of test seems to be critical before moving forward with an industry scale questionnaire modeling DSL.

Considering the full picture of this research, the following topics should be addressed moving forward: to formally evaluate SLang usability; to trace a language evolution plan that can help deal with variabilities in questionnaire specification practices; to investigate the development of a DSLs for questionnaire presentation modeling and make SLang executable to facilitate the questionnaire models creation. All of these issues can bring important contributions to MDSE, DSL development and survey methodology knowledge.

7
Bibliography

ARANGO, G. **A brief introductino to domain analysis**. SAC '94 Proceedings of the 1994 ACM Symposium on Applied Computing. Phoenix: ACM. 1994. p. 42-46.

BÈZIVIN, J. Model Driven Engineering: An Emerging Technical Space. In: LÄMMEL, R.; SARAIVA, J.; VISSER, J. **Generative and Transformational Techniques in Software Engineering**. Braga: Springer, 2005. p. 36-64.

BÈZIVIN, J.; GERBÉ, O. **Towards a Precise Definition of the OMG/MDA Framework.** IEEE International Conference on Automated Software Engineering. [S.l.]: IEEE. 2001. p. 273-280.

BEZIVIN, J. In search of a basic principle for model driven engineering. **Novatica/Upgrade**, v. 5, n. 2, p. 21-24, 2004.

BORODIN, A. V.; ZAVYALOVA, . V. **Ontology-Based Semantic Design of Survey Questionnaires**. Proceeding of the 19th conference of open inovations association (FRUCT). Jyvaskyla: IEEE. 2016. p. 10-15.

BROOKS JR, F. P. Language design as design. In: BERGIN JR., T. J.; GIBSON JR., R. G. **History of programming languages**. New York: ACM Press, v. II, 1996. p. 4 - 15.

BURDEN, H.; HELDAL, R.; WHITTLE, J. **Comparing and Contrasting Model-Driven Engineering at Three Large Companies**. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Torino: ACM. 2014.

CAMPAGNE, F. **The MPS Language Workbench**. Third Edition. ed. [S.l.]: [s.n.], 2016.

CORONEL, C.; MORRIS, S. **Database systems:** design, implementation and management. 13th. ed. [S.l.]: Cengage, 2017.

COTTON, F.; GILLMAN, D. . **Modeling the Statistical Process with Linked Metadata**. Proccedings of the 3rd International Workshop in Semantic Statistics. [S.l.]: [s.n.]. 2015.

COUPER, M. P. **Measuring survey quality in a CASIC environment**. Processdings ot the Survey Research Methods Section. [S.l.]: American Statistical Association. 1998. p. 41-49.

COUPER, M. P. Technology trends in survey data collection. **social Science Computer Review**, v. 23, n. 4, 2005.

CZECH, G.; MOSER, M.; PICHLER, J. **Best practices for domain-specific modeling. A systematic mapping study.** 44th Euromicro Conference on Software Engineering and Advanced Applications. Prague: IEEE. 2018. p. 137-145.

DATA DOCUMENTATION INNITIATIVE ALLIANCE. DDI Alliance Homepage. **DDI Alliance**, 2019. Available at: <https://.ddialliance.org>. Accessed on June 20, 2019.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: an annotated bibliography. **ACM SIGPLAN Notices**, New York, v. 35, n. 6, p. 26-36, June 2000.

ERDWEG, S. et al. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. **Computer Languages, Systems & Structures**, v. 44, n. A, p. 24-47, 2015.

EVANS, E. **Domain-driven design:** tackling complexity in the heart of doftware. [S.l.]: Addison Wesley, 2003.

FALBO, R. D. A.; GUIZZARDI, G.; DUARTE, K. C. **An Ontological Approach to Domain Engineering**. Processings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02). [S.l.]: ACM. 2002. p. 351-358.

FAVRÈ, J.-M.; NGUYEN, T. Towards a Megamodel to Model Software Evolution Through Transformations. **Eletronic Notes in Theoretical Computer Science**, 127, n. 3, 2005. 59-74.

FOWLER, M. Language Workbech. **Martin Fowler**, 2005. Available at: <http://martinfowler.com/articles/languageWorkbench.html>. Accessed on August 1st, 2019.

FOWLER, M. et al. **Patterns of enterprise application architecture**. [S.l.]: Addison-Wesley, 2003.

FOWLER, M.; PARSONS, R. **Domain-specific languages**. [S.l.]: Addison-Wesley, 2014.

FOWLER; MARTIN. A pedagogical framework for domain-specific languages. **IEEE Software**, v. 26, n. 4, p. 13-14, July/August 2009.

FRAKES, .; PRIETO-DIAZ, R.; FOX, C. DARE: Domain analysis and reuse environment. **Annals of Software Engineering**, v. 5, n. 1, p. 125-141, January 1998.

FRANK, U. **Some guidelines for the conception of domain-specific modeling languages**. Proceedings of the 4th International Workshop on Enterprise Modelling and Information Systems Architectures. Hamburg, Germany: [s.n.]. 2011. p. 93-106.

GIRALDO, D. et al. considerations about quality in model-driven engineering: current state and challenges. **Software Quality Journal**, v. 26, p. 685-750, 2018.

GOOGLE DEVELOPERS. Guide to app architecture. **Anroid developers**, 2019. Available at: <https://developer.android.com/jetpack/docs/guide>. Accessed on August 27, 2019.

GROVES, R. M. Three eras of survey research. **Public Opinion Quarterly**, v. 75, n. 5, p. 861-871, 2011.

GROVES, R. M. et al. **Survey Methodology**. 2nd. ed. New Jersey: John Wiley & Sons, 2009.

GSIM. GSIM and standards. **UNECE Statiswiki**, 2019. Available at: <https://statswiki.unece.org/display/gsim/GSIM+and+standards>. Accessed on June 1st, 2019.

HAUGEN, O.; PARASTOO, M. **A multi-dimentional framework for characterizing domain-specific languages**. Proceeding of the 7th OOPSLA Workshop on Domain Specific Modeling. [S.l.]: [s.n.]. 2007.

HEERING, J.; MERNIK, M. **Domain-Specific Languages in Perspective**. [S.l.]: [s.n.], 2007.

HUTCHINSON, J.; WHITTLE, J.; ROUNCEFIELD, M. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. **Science of Computer Programming**, v. 89, n. B, p. 144-161, September 2014.

IBGE. Census of Agriculture. **Brazilian Institute of Geography and Statistics**, 2017. Available at: <https://.ibge.gov.br/en/statistics/economic/agriculture-forestry-and-

fishing/21929-2017-2017-censo-agropecuario-en.html?=&t=o-que-e>. Accessed on April 5th, 2019.

JACKSON, E. K.; SZTIPANOVITS, J. **Towards a formal foundation for domain specific modeling languages**. Proccedings of the 6th ACM & IEEE International Conference on Embedded Software. [S.l.]: ACM. 2006.

JATAIN, A.; GOEL, S. Comparison of domain analysis methods in software reuse. **International Journal of Information Technology and Knowledge Management**, v. 2, n. 2, p. 347-352, December 2009.

KÜHNE, T. Matters of (meta-) modeling. **Software & Systems Modeling**, v. 5, n. 4, p. 369-385, December 2006.

KAHLAOUI, A.; ABRAN, A.; LEFEBVRE, É. Dsml success factors and their assessment criteria. **Metrics News**, v. 13, n. 1, p. 43-51, 2008.

KAHRAMAN, G.; BILGEN, S. A framework for qualitative assessment of domain-specific languages. **Software and Systems Modeling**, v. 14, n. 4, p. 1505–1526, October 2015.

KANG, K. C. et al. **Feature-Oriented Domain Analysis (FODA)**. Pittsburgh: Software Engineering Institute Carnegie Mellon University, 1990.

KARGE, R. **Integrated metadata-systems within statistical offices**. Tenth International Conference on Scientific and Statistical Database Management. Capri: [s.n.]. 1998. p. 216-219.

KARSAI, G. et al. **Design guidelines for domain specific languages**. Proceedings of the 9th OOPSLA workshop on domain-specific modeling. Florida: [s.n.]. 2009.

KELLY, S.; TOLVANEN, J.-P. **Domain-specific modeling:** enabling full code generation. New Jersey: John Wiley & Sons, 2008.

KÄRNA, J.; TOLVANEN, J.-P.; KELLY, S. **Evaluating the use of domain-specific modeling in practice**. Proceedings of DSM09. [S.l.]: [s.n.]. 2009.

KIM, C. H.; GRUNDY, J.; HOSKING, J. A suit of visual languages for model-driven development of statistical surveys and services. **Journal of Visual Languages and Computing**, Orlando, 26, n. 99, 2015.

KOLOVOS, D. S. et al. **Requirements for domain-specific languages**. Proceedings of the First ECOOP Workshop on Domain-Specific Program Development. Nates: [s.n.]. 2006.

LANDIN, P. J. The next 700 programming languages. **Communications of the ACM**, New York, v. 9, n. 3, p. 157-166, March 1966.

LISBOA, L. B. et al. A systematic review of domain analysis tools. **Information and Software Technology**, v. 52, n. 1, p. 1-13, 2010.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, New York, v. 37, n. 4, p. 316-344, December 2005.

MOHAGHEGHI, P. et al. An empirical study of the state of the practiceand acceptance of model-driven engineering in four industrial cases. **Empirical Software Engineering**, v. 18, n. 1, p. 89-116, February 2013.

MOHAGHEGHI, P.; DEHLEN, V. **Where is the proof? - A review of experiences from applyting MDE in industry**. European Conference on Model Driven Architecture - Foundations and Applications. [S.l.]: Springer. 2008. p. 432-443.

MOORE, D. S.; MCCABE, G. P.; CRAIG, B. A. **Introduction to the practice of statistics**. New York: W. H. Freeman and Company, 2009.

MUSSBACHER, G. et al. **The relevance of Model-driven engineering thirty years from now**. 17th International Conference ACM/IEEE - Conference on Model Driven Engineering Languages and Systems (MODELS). [S.l.]: [s.n.]. 2014.

NASCIMENTO, L. M. D. et al. **A systematic mapping study on domain-specific languages**. The Seventh International Conference on Software Engineering Advances (ICSEA 2012). Lisboa: IARIA XPS Press. 2012. p. 179-187.

NEIGHBORS, J. M. **Software construction using components**. Irvine: Department of Information and Computer Science University of California, 1980.

OPEN DATA KIT. Open Data Kit Home Page. **Open Data Kit**, 2019. Available at: <https://opendatakit.org>. Accessed on May 17, 2019.

PIETRO-DÍAZ, R. Domain analysis: an introduction. **Software Engineering Notes**, v. 15, n. 2, p. 47-54, April 1990.

QUALTRICS. Qualtrics Home Page. **Qualtrics**, 2019. Available at: <https://.qualtrics.com>. Accessed on May 16, 2019.

QUESTION. **English Oxford Dictionaries**. Available at: <https://en.oxforddictionaries.com/definition/question>. Accessed on April 11, 2019.

QUESTIONNAIRE. **English Oxford Dictionaries**. Available at: <https://en.oxforddictionaries.com/definition/questionnaire>. Accessed on April 11, 2019.

SARIS, W. E.; GALHOFER, I. N. **Design, Evaluation, and Analysis of Questionaires for Survey Research**. 2nd Edition. ed. Hoboken: John Wiley and Sons, Inc., 2014.

SEIDEWITZ, E. What models mean. **IEEE Software**, v. 20, n. 5, p. 26-32, 2003.

SELIC, B. The pragmatics of model-driven development, v. 20, n. 5, p. 19-25, 2003.

SENDAL, S.; KOZACSYNSKI, W. Model Transformation: The Heart and Soul of Model-Driven Software Development. **IEEE Software**, v. 20, n. 5, p. 42-45, September 2003.

SILVA, A. R. Model-driven engineering: A survey supported by the unified conceptual model. **Computer Languages, Systems & Structures**, v. 43, p. 139-155, October 2015.

SIMOS, M. A. **Organization domain modeling (ODM:** formalizing the core domain modeling life cycle. SSR'95: Proceeding of the 1995 Symposium on Software Reusability. [S.l.]: ACM. 1995. p. 196-205.

SPINELLIS, D. Notable design patterns for domain-specific languages. **The Journal of Systems and Software**, v. 56, p. 91-99, 2001.

STATISTICAL DATA AND METADATA EXCHANGE. SDMX. **SMDX Community**, 2019. Available at: <https://sdmx.org/>. Accessed on June 1st, 2019.

STATISTICS NETHERLANDS. Blaise Home Page. **Blaise**, 2019. Available at: <https://www.blaise.com>. Accessed on June 1st, 2019.

STREMBECK, M.; ZDUN, U. An apporach for the systematic development of domain-specific languages. **Software: practice and experience**, v. 39, n. 15, p. 1253-1292, September 2009.

SURVEYMONKEY. SurveyMonkey Home Page. **SurveyMonkey**, 2019. Available at: <https://pt.surveymonkey.com/>. Accessed on May 15, 2019.

THIBAULT, S. A.; MARLET, R.; CONSEL, C. Domain-specific languages: from design to implementation application to video device drivers generation. **Transactions on software engineering**, v. 25, n. 3, p. 363-377, May/June 1999.

TRACZ, W. DSSA (domain-Specifc Software Architecture): pedagogical example. **ACM SIGSOFT Software Engineering Notes**, v. 20, n. 3, p. 49-61, July 1995.

UNECE. GSIM Communication Paper. **Statswiki**, 2019. Available at: <https://statswiki.unece.org/display/gsim/GSIM+Communication+Paper>. Accessed on June 17, 2019.

UNITED STATED CENSUS BUREAU. Census and Survey Processing System. **United States Census Bureau Website**, 16 may 2019. Available at: <https://www.census.gov/data/software/cspro.html>. Accessed on May 16, 2019.

UNITED STATES GENERAL ACCOUNTING OFFICE. **Developing and Using Questionnaires**. [S.l.]. 1993. (GAO/PEMD-10.1.7).

VARDAKI, M.; PAPAGEORGIOU, H. **An integrated metadata model for statistical data collection and processing**. 16th International Conference on Scientific and Statistical Database Management. [S.l.]: IEEE. 2004.

VASUDEVAN, N. Comparative study of DSL tools. **Eletronic Notes in Theoretical Computer Science**, v. 264, n. 5, p. 103-212, 2011.

VOELTER, M. et al. **DSL Engineering**. [S.l.]: CreateSpace Independent Publishing Platform, 2013.

WASILEWSKI, M.; HASSELBRING, W.; NOWOTKA, D. **Defining requirements on domain-specific languages in model-driven software engineering of safety-critical systems**. Software Engineering 2013 Workshopband. [S.l.]: Gesellschaft für Informatik. 2013. p. 467-482.

WEGELER, T. et al. **Evaluating the Benefits of Using Domain-Specific Modeling languages - an experience report**. Proceedings of the 2013 ACM workshop on domain-specific modeling. Indianapolis: ACM. 2013. p. 7-12.

WEISS, D. M.; LAI, C. T. R. **Software Product-Line Engineering:** A family-Based Software development Process. [S.l.]: Addison-Wesley, 1999.

WHITTLE, J.; HUTCHINGSON, J.; ROUCEFIELD, M. The state of practive in model-driven engineering. **IEEE Software**, v. 31, n. 1, p. 79-85, May-June 2014.

WIKIPEDIA. Data Documentation Initiative. **Wikipedia**, 2019. Available at: <Data Documentation Initiative>. Accessed on June 13, 2019.

WIKIPEDIA. Empiricism. **Wikipedia**, 2019. Available at: <https://en.wikipedia.org/wiki/Empiricism>. Accessed on August 18, 2019.

WIKIPEDIA. Model transformation. **Wikipedia**, 2019. Available at: <https://en.wikipedia.org/wiki/Model_transformation>. Accessed on June 6, 2019.

WIKIPEDIA. SDMX. **Wikipedia**, 2019. Available at: <https://en.wikipedia.org/wiki/SDMX>. Accessed on June 13, 2019.

ZOHO. ZoHo Home Page. **ZoHo**, 2019. Available at: <https://www.zoho.com/survey/>. Accessed on May 16, 2019.

Appendix I

Slang model for the 2015 PeNSE Survey school questionnaire.

```
Survey: National Student Health Survey  (PENSE2015)
  description:
  version: 1 !! allows to control changes in the survey specification
  constants:
    min_wage: number 778
      empty_value: number 0
      skipped_value: number 9

  Objects of interest:
    alias: school
    school_id: number
    school_type: number
    school_uf: number
    school_city: number
    school_type: number
    school_situation: number
    school_administration: number
    school_public_administration_scope: number

  Theme: School information – T01
    Question Set:
      Question: 1.1.1 - School information
        Item:
          description: Visit date:
            save at: date visit_date
        Item:
            description: School:
            save at: domain school_id
            read from: schools

  Theme: School environment – T02
    description: The following questions regard the school environment and should be
    answered by the school principal, coordinator or the person responsible for the school.
    Question Set:
      Question: 2.1.1 – What is your role at the school?
        Item:
            save at: domain interviewee_role
            1: Principal
            2: Coordinator
            3: Administrator
            4: Secretary
            5: Teacher
            6: Other
       Question: 2.1.2 – What are the school shifts?
          Item:
```

save at: domain school_shifts
1: Morning
2: Intermediate (part morning, part afternoon)
3: Afternoon
4: Evening

Question: 2.1.3 – Does the school provides services full-time?
Item:
save at: domain school_fulltime
1: Yes
2: No

Question: 2.1.4 – Is the school a boarding school?
Item:
save at: domain school_boarding
1: Yes
2: No

Question: 2.1.5 – Which levels does the school supports?
Item:
save at: domain school_levels
1: Preschool
2: K-12
3: High school
4: Special programs

Question: 2.1.6 – What is the number of enrroled students?
Item:
save at: domain school_student_qtd
1: Up to 50 students
2: From 51 to 100 students
3: From 101 to 500 students
4: From 501 to 1000 students
5: More than 1000 students

Question: 2.1.7 – What is the total number of classrooms?
Item:
save at: domain school_room_qtd
1: Up to 10 classrooms
2: From 11 to 20 classrooms
3: From 21 to 30 classrooms
4: From 31 to 40 classrooms
5: From 41 to 50 classrooms
6: More than 51 classrooms

Question: 2.1.8 – How much is school tuition at the K-12 9th grade?
information: Minimum wage in 01-01-2015 = R$ {min_wage},00
visibility: school_administration == 2
Item:
save at: domain school_tuition
1: Up to R$ 394,00
2: More than R$ 394,00 up to R$ 788,00
3: More than R$ 788,00 up to R$ 1.576,00
4: More than R$ 1.576,00 up to R$ 3.152,00
5: More than R$ 3.152,00 up to R$ 6.304,00
6: More than R$ 6.304,00

Question: 2.1.9 – Does the school has a library that is in condition of use?
Item:
save at: domain school_library
1: Yes
2: No
3: There is no library

Question: 2.1.10 – Does the school has a room or computer lab that is in condition of use?

Item:

save at: domain school_computer_lab

1: Yes

2: No

3: There is no room or computer lab

Question: 2.1.11 – How many school computers (desktops, laptops, notebooks, netbooks, tablets) in conditions of use are available for the students in classrooms or in specific computer rooms?

Item:

save at: domain school_computer_qtd

1: Up to 10 computers

2: From 11 to 20 computers

3: From 21 to 30 computers

4: From 41 to 50 computers

5: More than 50 computers

Question: 2.1.12 – Does students can access the internet from school computers?

Item:

save at: domain school_internet

1: Yes

2: No

Question: 2.1.13 – Does the school has multimedia/communications room in conditions of use (Examples: television, videocassette, DVD, projectors, etc.)?

Item:

save at: domain school_multimedia

1: Yes

2: No

3: There is no multimedia/communications room

Question: 2.1.14 – Does the school has a school council?

Item:

save at: domain school_council

1: Yes

2: No

Question: 2.1.15 – How frequently does the school council holds meetings?

visibility: school_council == 1

Item:

save at: domain school_council_freq

1: There are no scheduled meetings

2: From 1 to 3 times a year

3: From 4 to 6 times a year

4: From 7 to 9 times a year

5: From 10 to 12 times a year

6: More than 12 times a year

Question: 2.1.16 – Does the school remains open during weekends so that the community can use its installations?

Item:

save at: domain school_weekend_activities

1: Yes

2: No

Question: 2.1.17 – Are the activities developed during weekends shared with planned with the community participation?

visibility: school_weekend_activities == 1

Item:

save at: domain school_weekend_activities_planning

1: Yes

2: No

Theme: Sports practice – T03
Question Set:
Question: 3.1.1 – Does the school has sports quarts in conditions of use?
Item:
save at: domain school_sports_quart
1: Yes
2: No
3: There is no sports quart
Question: 3.1.2 – How many sports quarts, in conditions of use, does the school has?
visibility: school_sports_quart == 1
Item:
save at: domain school_sports_quart_qtd
1: 1
2: 2
3: 3 or more
Question: 3.1.3 – How many of the sport quarts, in conditions of use, are sheltered?
visibility: school_sports_quart == 1
Item:
save at: domain school_sports_quart_sheltered
1: None
2: 1
3: 2
4: 3 or more
5: All
Question: 3.1.4 – Does the school has athletics track in conditions of use?
Item:
save at: domain school_sports_athletics_track
1: Yes
2: No
3: There is no athletics track
Question: 3.1.5 – Does the school has pools in conditions of use?
Item:
save at: domain school_sports_pool
1: Yes
2: No
3: There is no pool
Question: 3.1.6 – Is the school patio used for instructor lead regular physical activities?
Item:
save at: domain school_sports_patio
1: Yes
2: No
3: There is patio
Question: 3.1.7 – Does the school has sports and games material in conditions of use?
Item:
save at: domain school_sports_material
1: Yes
2: No
3: There is no sports and game material
Question: 3.1.8 – Does the school has locker rooms in conditions of use for the students?
Item:
save at: domain school_sports_locker_rooms
1: Yes
2: No
3: There is no locker room

Question: 3.1.9 – Does the school has separate locker rooms for boys and girls?
visibility: school_sports_locker_rooms == 1
Item:
save at: domain school_sports_locker_rooms_separate
1: Yes
2: No
3: There are no separate locker rooms
Question: 3.1.10 – Does the school offers sports activities outside the regular hours?
Item:
save at: domain school_sports_activities
1: Yes
2: No
3: There are no separate locker rooms
Question: 3.1.11 – Does the school takes part in competitions between schools?
Item:
save at: domain school_sports_competitions
1: Yes
2: No
Question: 3.1.12 – Does the school organized competitions between classes of shifts?
Item:
save at: domain school_sports_competitions_internal
1: Yes
2: No

Theme: Accessibility – T04
Question Set:
Question: 4.1.1 – Does the school has students with deficiency or global development deficiency?
Item:
save at: domain school_accessibility_deficiency
1: Yes
2: No
Question: 4.1.2 – What type(s) of deficiency?
visibility: school_accessibility_deficiency == 1
Item:
description: Intellectual deficiency
save at: number school_accessibility_deficiency_intelectual
Item:
description: Autism spectrum disorder
save at: number school_accessibility_deficiency_autism
Item:
description: Mental and behavioral disorders
save at: number school_accessibility_deficiency_mental
Item:
description: Physical deficiency
save at: number school_accessibility_deficiency_physical
Item:
description: Hearing deficiency
save at: number school_accessibility_deficiency_hearing
Item:
description: Visual deficiency
save at: number school_accessibility_deficiency_visual
Item:
description: Multiple deficiencies
save at: number school_accessibility_deficiency_multiple
Item:
description: Others

save at: number school_accessibility_deficiency_other

Question: 4.1.3 – Does the school offers physical activities adapted for students with disabilities?
visibility: school_acessibility_deficiency == 1
Item:
save at: domain school_accessibility_sports
1: Yes
2: No

Question: 4.1.4 – Does school infrastructure provides accessibility for students with disabilities?
visibility: school_acessibility_deficiency == 1
Item:
save at: domain school_accessibility_infrastructure
1: Yes
2: No

Question: 4.1.5 – What kind of infrastructure aspects are available to guarantee accessibility for students with disabilities?
visibility: school_accessibility_deficiency == 1 &&
school_acessibility_infrastructure == 1
Item:
description: Ramps
save at: number school_accessibility_infrastructure_ramps
Item:
description: Adequate space for locomotion
save at: number school_accessibility_infrastructure_locomotion
Item:
description: Adequate furniture for students with disabilities
save at: number school_accessibility_infrastructure_furniture
Item:
description: Adequate toilets
save at: number school_accessibility_infrastructure_toilet

Theme: Nutrition – T05
Question Set:
Question: 5.1.1 – Does the school offers meals to students?
Item:
save at: domain school_nutrition_meals
1: Yes
2: No

Question: 5.1.2 – Does the school offers meals for which shifts?
visibility: school_nutrition_meals == 1
Item:
description: Morning
save at: number school_nutrition_shift_morning
Item:
description: Intermediate
save at: number school_nutrition_shift_intermediate
Item:
description: Afternoon
save at: number school_nutrition_shift_afternoon
Item:
description: Evening
save at: number school_nutrition_shift_evening

Question: 5.1.3 – Does the school has a kitchen in conditions of use?
Item:
save at: domain school_nutrition_kitchen
1: Yes

2: No
3: There is no kitchen

Question: 5.1.4 – Does the school has a dining hall in conditions of use?
Item:
save at: domain school_nutrition_dining_hall
1: Yes
2: No
3: There is no dining hall

Question: 5.1.5 – Does the school has a vegetable garden?
Item:
save at: domain school_nutrition_vegetable_garden
1: Yes
2: No

Theme: Basic sanitation and hygiene - T06
Question Set:

Question: 6.1.1 - Does the school offers drinking water to students?
Item:
save at: domain school_hygiene_water
1: Yes
2: No
3: There is no water

Question: 6.1.2 – In the past 12 months has the school water quality been tested?
visibility: school_hygiene_water == 1
Item:
save at: domain school_hygiene_water_quality
1: Yes
2: No

Question: 6.1.3 – What is the school main source of drinking water?
visibility: school_hygiene_water == 1
Item:
save at: domain school_hygiene_water_source
1: Public facility
2: Well or spring
3: Rainwater (cistern)
4: Weir, lake or river
5: Other source

Question: 6.1.4 – Does the school has toilets in conditions of use?
Item:
save at: domain school_hygiene_toilets
1: Yes
2: No
3: There is no bathroom

Question: 6.1.5 – Does the school has separate bathrooms for boys and girls?
visibility: school_hygiene_toilets == 1
Item:
save at: domain school_hygiene_toilets_separate
1: Yes
2: No

Question: 6.1.6 – Does the school offers toilet paper in its toilets?
visibility: school_hygiene_toilets == 1
Item:
save at: domain school_hygiene_toilets_paper
1: Yes
2: No

Question: 6.1.7 – Does the school has a sink or lavatory where students can wash hands after using the toilet or before meals?

Item:
save at: domain school_hygiene_sink
1: Yes
2: No
3: There is no sink or lavatory

Question: 6.1.8 – Does the school offers soap for students to wash hands after using the toilet or before meals?

Item:
save at: domain school_hygiene_soap
1: Yes
2: No

Question: 6.1.9 – How frequent is the trash collected?

Item:
save at: domain school_hygiene_trash
1: There is no weekly collection
2: 1 to 2 days a week
3: 3 to 4 days a week
4: 5 to 6 days a week
5: Everyday

Theme: Security – T07

Question Set:

Question: 7.1.1 – In the past 12 months how frequently the school neighborhood was considered risky in terms of violence (theft, robbery, assault, gun firing, substance abuse, homicide)?

Item:
save at: domain school_violence_freq
1: Never
2: Rarely
3: Sometimes
4: Most of the time
5: All the time

Question: 7.1.2 – In the past 12 months, did the school suspend or interrupted class for safety reasons because of violence?

Item:
save at: domain school_violence_interruptions
1: Never
2: Once
3: One time
4: 2 to 4 times
5: 5 times or more

Theme: Health policy – T08

Question Set:

Question: 8.1.1 – Does the school has any type of committee responsible for defining or coordinating actions and activities related to health?

Item:
save at: domain school_health_committee
1: Yes
2: No

Question: 8.1.2 – Has the school joined the Health in School Program?

Item:
save at: domain school_health_hsp
1: Yes
2: No

Question: 8.1.3 – Does the school implements the actions prescribed in the Health School Program?

visibility: school_health_hsp == 1
Item:
save at: domain school_health_hsp_impl
1: Yes
2: No

Question: 8.1.4 – Does the school implements the More Education Program actions?
Item:
save at: domain school_health_mep
1: Yes
2: No

Question: 8.1.5 – Does the school implements actions together with health teams, health family teams or basic health attentions teams?
Item:
save at: domain school_health_actions
1: Yes
2: No

Question: 8.1.6 – Does the school keeps records of student vaccine shots?
Item:
save at: domain school_health_vaccine
1: Yes
2: No

Question: 8.1.7 – Does the school keeps first aid materials in an adequate place?
Item:
save at: domain school_health_first_aid
1: Yes
2: No
3: There is no first aid materials

Question: 8.1.8 – Does the school knows about teachers smoking in the school premises?
Item:
save at: domain school_health_teacher_smoking
1: Yes
2: No

Question: 8.1.9 – Does the school knows about students smoking in the school premises?
Item:
save at: domain school_health_student_smoking
1: Yes
2: No

Question: 8.1.10 – Does the school has any written rule or policy prohibiting the usage of tobacco on its premises?
Item:
save at: domain school_health_tobacco_policy
1: Yes
2: No

Question: 8.1.11 – Does the school has any written rule or policy prohibiting the usage of alcohol on its premises?
Item:
save at: domain school_health_alcohol_policy
1: Yes
2: No

Question: 8.1.12 – Does the school has any written rule or policy prohibiting the usage of drugs on its premises?
Item:
save at: domain school_health_drugs_policy
1: Yes
2: No

Question: 8.1.13 – Does the school has any written rule or policy prohibiting bullying on its premises?

Item:

save at: domain school_health_bullying_policy

1: Yes

2: No

Question: 8.1.14 – Does the school has any written rule or policy prohibiting fights on its premises?

Item:

save at: domain school_health_fights_policy

1: Yes

2: No

Question: 8.1.15 – Does the school has any written rule or policy prohibiting students physical punishment on its premises?

Item:

save at: domain school_health_physical_punishment_policy

1: Yes

2: No

Appendix II

Slang model for the 2015 PeNSE survey school questionnaire dictionary

```
Dictionary for: National Student Health Survey  (PENSE2015)
  Measurements
    school_id
    code: V0001
    precision: 8
    scale: 0

    school_type
    code: V0002
    type: number
    precision: 2
    scale: 0
    rule: school.type

    school_uf
    code: V0003
    type: number
    precision: 7
    scale: 0
    rule: school.uf

    school_city
    code: V0004
    type: number
    precision: 7
    scale: 0
    rule: school.city

    school_situation
    code: V0005
    type: number
    precision: 1
    scale: 0
    rule: school.situation

    school_administration
    code: V0006
    type: number
    precision: 1
    scale: 0
    rule: school.administration

    school_public_administration_scope
    code: V0007
    type: number
    precision: 1
```

scale: 0
rule: school.public_administration_scope

interviewee_role
code: E01P29
type: domain
precision: 1
scale: 0

school_shifts
code: E01P30
type: domain
precision: 1
scale: 0

school_fulltime
code: E01P31
type: domain
precision: 1
scale: 0

school_boarding
code: E01P32
type: domain
precision: 1
scale: 0

school_levels
code: E01P03a
type: domain
precision: 1
scale: 0

school_student_qtd
code: E01P02a
type: domain
precision: 1
scale: 0

school_room_qtd
code: E01P04a
type: domain
precision: 1
scale: 0

school_tuition
code: E01P01
type: domain
precision: 1
scale: 0

school_library
code: E01P05a
type: domain
precision: 1
scale: 0

school_computer_lab
code: E01P06a
type: domain
precision: 1
scale: 0

school_computer_qtd
code: E01P33
type: domain
precision: 1
scale: 0

school_internet
code: E01P09
type: domain
precision: 1
scale: 0

school_multimedia
code: E01P10a
type: domain
precision: 1
scale: 0

school_council
code: E01P23
type: domain
precision: 1
scale: 0

school_council_freq
code: E01P24a
type: domain
precision: 1
scale: 0

school_weekend_activities
code: E01P34
type: domain
precision: 1
scale: 0

school_weekend_activities_planning
code: E01P35
type: domain
precision: 1
scale: 0

school_sports_quart
code: E01P15a
type: domain
precision: 1
scale: 0

school_sports_quart_qtd
code: E01P16a

type: domain
precision: 1
scale: 0

school_sports_quart_sheltered
code: E01P17a
type: domain
precision: 1
scale: 0

school_sports_athletics_track
code: E01P18a
type: domain
precision: 1
scale: 0

school_sports_pool
code: E01P20
type: domain
precision: 1
scale: 0

school_sports_patio
code: E01P19
type: domain
precision: 1
scale: 0

school_sports_material
code: E01P36
type: domain
precision: 1
scale: 0

school_sports_locker_rooms
code: E01P21
type: domain
precision: 1
scale: 0

school_sports_locker_rooms_separate
code: E01P37
type: domain
precision: 1
scale: 0

school_sports_activities
code: E01P22
type: domain
precision: 1
scale: 0

school_sports_competitions
code: E01P38
type: domain
precision: 1
scale: 0

school_sports_competitions_internal
code: E01P39
type: domain
precision: 1
scale: 0

school_accessibility_deficiency
code: E01P40
type: domain
precision: 1
scale: 0

school_accessibility_deficiency_intelectual
code: E01P41a
type: number
precision: 1
scale: 0

school_accessibility_deficiency_autism
code: E01P41b
type: number
precision: 1
scale: 0

school_accessibility_deficiency_mental
code: E01P41c
type: number
precision: 1
scale: 0

school_accessibility_deficiency_physical
code: E01P41d
type: number
precision: 1
scale: 0

school_accessibility_deficiency_hearing
code: E01P41e
type: number
precision: 1
scale: 0

school_accessibility_deficiency_visual
code: E01P41f
type: number
precision: 1
scale: 0

school_accessibility_deficiency_multiple
code: E01P41f
type: number
precision: 1
scale: 0

school_ accessibility _deficiency_other
code: E01P41g

type: number
precision: 1
scale: 0

school_accessibility_sports
code: E01P42
type: domain
precision: 1
scale: 0

school_accessibility_infrastructure
code: E01P43
type: domain
precision: 1
scale: 0

school_accessibility_infrastructure_ramps
code: E01P44a
type: number
precision: 1
scale: 0

school_accessibility_infrastructure_locomotion
code: E01P44b
type: number
precision: 1
scale: 0

school_accessibility_infrastructure_furniture
code: E01P44c
type: number
precision: 1
scale: 0

school_accessibility_infrastructure_toilet
code: E01P44d
type: number
precision: 1
scale: 0

school_nutrition_meals
code: E01P45
type: domain
precision: 1
scale: 0

school_nutrition_shift_morning
code: E01P46a
type: number
precision: 1
scale: 0

school_nutrition_shift_intermediate
code: E01P46b
type: number
precision: 1
scale: 0

school_nutrition_shift_afternoon
code: E01P46c
type: number
precision: 1
scale: 0

school_nutrition_shift_evening
code: E01P46d
type: number
precision: 1
scale: 0

school_nutrition_kitchen
code: E01P47
type: domain
precision: 1
scale: 0

school_nutrition_dining_hall
code: E01P48
type: domain
precision: 1
scale: 0

school_nutrition_vegetable_garden
code: E01P49
type: domain
precision: 1
scale: 0

school_hygiene_water
code: E01P50
type: domain
precision: 1
scale: 0

school_hygiene_water_quality
code: E01P51
type: domain
precision: 1
scale: 0

school_hygiene_water_source
code: E01P52
type: domain
precision: 1
scale: 0

school_hygiene_toilets
code: E01P53
type: domain
precision: 1
scale: 0

school_hygiene_toilets_separate
code: E01P54

type: domain
precision: 1
scale: 0

school_hygiene_toilets_paper
code: E01P55
type: domain
precision: 1
scale: 0

school_hygiene_sink
code: E01P56
type: domain
precision: 1
scale: 0

school_hygiene_soap
code: E01P57
type: domain
precision: 1
scale: 0

school_hygiene_trash
code: E01P58
type: domain
precision: 1
scale: 0

school_violence_freq
code: E01P25
type: domain
precision: 1
scale: 0

school_violence_interruptions
code: E01P59
type: domain
precision: 1
scale: 0

school_health_committee
code: E01P60
type: domain
precision: 1
scale: 0

school_health_hsp
code: E01P61
type: domain
precision: 1
scale: 0

school_health_hsp_impl
code: E01P62
type: domain
precision: 1
scale: 0

school_health_mep
code: E01P63
type: domain
precision: 1
scale: 0

school_health_actions
code: E01P64
type: domain
precision: 1
scale: 0

school_health_vaccine
code: E01P65
type: domain
precision: 1
scale: 0

school_health_first_aid
code: E01P66
type: domain
precision: 1
scale: 0

school_health_teacher_smoking
code: E01P67
type: domain
precision: 1
scale: 0

school_health_student_smoking
code: E01P27
type: domain
precision: 1
scale: 0

school_health_tobacco_policy
code: E01P28a
type: domain
precision: 1
scale: 0

school_health_alchool_policy
code: E01P68
type: domain
precision: 1
scale: 0

school_health_drugs_policy
code: E01P69
type: domain
precision: 1
scale: 0

school_health_bullying_policy
code: E01P70

type: domain
precision: 1
scale: 0

school_health_fights_policy
code: E01P71
type: domain
precision: 1
scale: 0

school_health_physical_punishment_policy
code: E01P72
type: domain
precision: 1
scale: 0