DSL Based Approach for Building Model-Driven Questionnaires

Luciane Calixto de Araujo[1], Marco A. Casanova[2](✉), Luiz André P. P. Leme[3], and Antônio L. Furtado[2]

[1] Brazilian Institute of Geography and Statistics, Rio de Janeiro, Brazil
luciane.araujo@ibge.gov.br

[2] Department of Informatics, PUC-Rio, Rua Marquês de São Vicente, 225, Rio de Janeiro, Brazil
{casanova,furtado}@inf.puc-rio.br

[3] Institute of Computing, Federal Fluminense University, Av. Gal. Milton Tavares de Souza, s/n, Niterói, Brazil
lapaesleme@ic.uff.br

Abstract. Surveys are pervasive in the modern world, with its usage ranging from the field of customer satisfaction measurement to global economic trends tracking. Data collection is at the core of survey processes and, usually, is 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. 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. This article proposes a Domain Specific Language (DSL) for modeling questionnaires, presents a prototype, and evaluates DSL use as a strategy to reduce the gap between survey domain experts and software developers, improve reuse, eliminate redundancy, and minimize rework.

Keywords: Survey questionnaires · Domain-specific languages · Model-driven software engineering

1 Introduction

A survey is a systematic method for collecting data about (a sample of) entities to construct quantitative descriptors of the attributes of a larger population of which the entities are members. The usage of a questionnaire is, by far, the most common data collection strategy [1].

Highly influenced by recent information technology advances, developing software to support a questionnaire-based survey seems to be an ordinary software engineering task. After all, questionnaires are forms, for which a large number of different solutions and development strategies exist. However, as survey scales in size or complexity, this ordinary task becomes daunting. A questionnaire can comprise hundreds of variables intertwined in a complex web of data quality controls implemented to guarantee that each question is fully understood and adequately answered. Hence, underestimating questionnaire design complexity is a common flaw that directly impacts survey quality [1].

In engineering, complexity is frequently handled by raising the level of abstraction. In particular, model-driven software engineering (MDSE) aims at raising computer language abstraction further by making models first-class citizens in the software development process. This article proposes the usage of a model-driven approach for designing complex questionnaires. Specifically, it proposes a Domain Specific Language (DSL) for modeling questionnaires, presents a prototype, and evaluates the use of the DSL as a strategy to reduce the gap between survey domain experts and software developers, improve reuse, eliminate redundancy and minimize rework.

In more detail, the article first describes a domain analysis that resulted in a model for the structure of questionnaires, including elements that allow the modeling of questionnaire data consistency and integrity rules, as well as the specification of behavioral aspects of the questionnaire required to capture navigation flow.

Next, based on the proposed model, the article describes the design of a prototype domain-specific language (DSL) for modeling complex questionnaires, called SLang, and its implementation using the MPS projectional language workbench. The SLang design process and the main decisions provide insights on how model-driven DSL approaches can be applied to real-world problems.

Finally, the article covers the evaluation of SLang, providing a good picture of the proposed solution in practice. In this evaluation, real-world questionnaires modeled with SLang were deployed in a complete setting, including both the SInterviewer app and a backend responsible for centralizing collected data, with real end-users executing mock interviews.

SLang prototyping has been previously described in [2]. This article contains additional insights on the survey questionnaires domain and how the domain analysis was performed, additional information on the SLang design, with improved examples, and details of the language evaluation process, including an IT environment that is closer to reality.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 covers the domain analysis and the questionnaire model. Section 4 describes the SLang design decisions and its main features. Section 5 details the implementation and evaluation of SLang. Section 6 presents conclusions and further developments.

2 Related Work

2.1 Survey Questionnaires

Survey questionnaires are part of a broader context in which a survey is planned and executed and produces results based on the analysis of collected data. As such, the survey process comprises a series of activities, executed in three stages: survey planning, survey execution, and survey data publishing (Fig. 1).

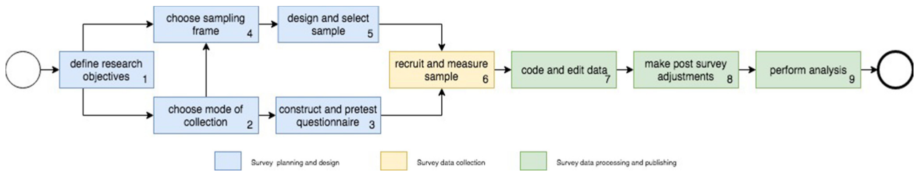
define research objectives 1
choose mode of collection 2
construct and pretest questionnaire 3
choose sampling frame 4
design and select sample 5
recruit and measure sample 6
code and edit data 7
make post survey adjustments 8
perform analysis 9
Survey planning and design
Survey data collection
Survey data processing and publishing

Fig. 1. Survey process [3].

From an IT point of view, the questionnaire has a role in each of the survey process phases. First, it must be created during the survey planning and design phase. During the questionnaire creation, metadata (data about the collected data), data (actual collected data), and paradata (data about the data collection process) are defined. Once the questionnaire has been created, the issue becomes how to use that questionnaire to support and control interviews during the data collection phase. Considering a reality where software is mandatory, supporting the interview means transforming the defined questionnaire into 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.

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

2.2 Survey Models

A few existing survey models exist in the literature, either in the form of standards, developed and maintained by entities related to the statistical survey community, or proposed as part of research endeavors.

Survey standards include GSIM, DDI, SDMX. GSIM is the first internationally endorsed reference framework of information objects, enabling generic descriptions of the definition, management, and use of data and metadata throughout the statistical production process [4]. DDI is an international standard for describing survey metadata, including questionnaires, statistical data files, and social sciences study-level information [5]. SDMX aims at standardizing the mechanisms and processes for the exchange of statistical data and metadata among international organizations and their member countries [6].

Since the goal of this paper is to provide a DSL-based approach for questionnaire design, GISM and DDI are relevant inputs for the domain analysis and parallel the development of the model proposed in this work.

Borodin and Zavyalova [7] described an ontology for survey questionnaires. The proposed ontology models questionnaires with simple navigation rules. However, it lacks support for grouping questions by theme, validating answers, and creating objects of interest, among other requirements for complex questionnaires. Still, the ontology is a useful reference for modeling questionnaires.

Table 1. Questionnaire related requirements for survey IT solutions.

Survey process phase	Requirement
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 the 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

2.3 Questionnaire Design Tools

Questionnaire design and data collection tools can be divided into Web-based survey tools and frameworks. Web-based survey tools work by allowing the user to create a questionnaire that will be distributed and answered through the Web. Google Forms

(GF), SurveyMonkey (SM), Zoho (ZH), and Qualtrics (QT) are examples of Web-based survey tools. CSPro (CS), developed by the US Census Bureau, Blaise (BL), developed by Netherlands Statistics, and Open Data Kit (ODK) are examples of frameworks. They usually include at least a questionnaire design tool and a data collection tool [8].

2.4 Domain Specific Language

DSLs are languages tailored to a specific application domain that offer substantial gains in expressiveness and ease of use in their application domain, when compared with general-purpose programming languages [9]. DSLs are almost as old general-purpose languages (GPLs) and have been applied to domains ranging from bioinformatics through robotics, including Web applications, embedded systems, low-level software, control systems, parallel computing, simulation, data-intensive apps, real-time systems, security, education, and networks [10].

In the domain of complex survey questionnaires, the use of DSLs for questionnaire design is not new. Kim et al. [11] developed the Survey Design Language (SDL) and its supporting tool, SDLTool. SDL consists of a set of domain-specific visual languages. Each language in the set is designed to model a specific aspect of statistical surveys, providing high-level and low-level modeling facilities capable of matching expert cognitive models for statistical surveys. The SDLTool was the environment that tied together the different DSLs aspects through visual modeling of survey resources, the design of statistical survey elements, running modeled surveys on target population datasets, and providing visualization support features [2].

The 2013 Language Workbench Challenge (LWC) assignment consisted of developing a DSL for questionnaires, which had to be rendered in an interactive GUI that reacted to user input and had to store the answers of 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 proposed languages offered basic questionnaire functionality but lacked primordial features, such as the possibility of specifying questionnaire navigation, complex validation rules, and triggers, among others [12].

Zhou, Goto, and Cheng [13] presented QSL, a language to specify questionnaire systems that includes some aspects of questionnaire design. Still, it is not clear whether QSL supports a questionnaire complex logic for navigation flow and question presentation. Also, the dynamic creation of objects of interest is not mentioned. Finally, QSL presents a tight coupling between questionnaire presentation and modeling [2].

None of the aforementioned DSLs were designed having questionnaire specification as its main goal. As such, each of them has shortcomings. SDL avoids completely questionnaire specification having its focus on survey methodology and data analysis. 2013 LWC languages were focused on demonstrating language workbenches potential and provided limited support for questionnaire specification. Finally, QSL had its focus on e-questionnaire systems with limited support for questionnaire specification [2].

3 Complex Survey Questionnaire Domain Analysis and Model

3.1 Questionnaire Domain Analysis

Domain analysis is the activity of identifying the objects and operations of a class of similar systems in a problem domain [14]. It involves the identification, acquisition, and analysis of domain knowledge to be reused in software specification and construction [15]. Through domain analysis, information about the domain is identified, collected, organized, and represented. The domain analysis may be based upon: the study of existing systems and their development histories; knowledge captured from domain expert; or the characteristics of emerging technology within the domain [16].

Researchers and practitioners proposed multiple ways of doing a domain analysis, each of them focused on specific goals. Although domain analysis for DSL design and implementation is in its beginnings, there is enough clarity on the general activities to be performed: domain scoping, data collection, data analysis, classification, and domain model evaluation [17].

In this work, the analysis of the domain of complex questionnaires was performed in 4 stages: scope definition, data collection, data analysis and classification, domain modeling. The domain model was evaluated through its usage and suitability in SLang design.

The scope of the domain of complex questionnaires was defined as the universe of statistical surveys whose data collection step has the following characteristics: information is gathered primarily by posing questions to people; 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; information is collected from (a sample of) the population to be described [3].

Considering these characteristics of the universe of statistical surveys, data collection included: survey planning and document specification; statistical survey data collection software documents; data collection software source code; and statistical survey standards experts' interviews [8]. Collected data was processed through an iterative process that included analyzing a piece of domain data and evolving the domain model.

3.2 Questionnaire Model

As collected data was analyzed, domain concepts were extracted, and their attributes, relationships, and pertaining information were documented. The domain model was organized into two main components: the structural model and the behavioral model. The structural model comprises metadata, data, and paradata showing the concepts and their relationships of each area. The behavioral model details the questionnaire navigation flow specification and state evolution.

Metadata Structural Model. Metadata concepts define the questionnaire structure, including consistency and integrity rules, to be applied during data collection. Figure 2 presents the metadata structural model. This model evolved continuously together with the domain understanding and knowledge consolidation.

Survey and *Dictionary* are the model root concepts and tie together the metadata structure. *Survey* is directly related to the *Theme*, *QuestionSet*, *Question*, and *QuestionItem* hierarchy, which hold the questionnaire actual content in the form of question stating, answer options, etc. *Theme*, *QuestionSet*, *Question*, and *QuestionItem* are also *NavigationItems* and are responsible for providing information for navigation flow and questionnaire state controls.

Measurements define the pieces of data that a questionnaire aims at gathering. *Measurements* can be related to a *QuestionItem* in a one to one relationship or stand by themselves (that is the case of *CalculatedMesurements*). A survey *Dictionary* is basically a measurement list. A detailed account of the aforementioned domain concepts can be found in [2, 8].

It is important to single out the modeling of *ObjectOfInterest* relationships. Each *Survey* is related to one or more "objects of interest". An *ObjectOfInterest* is a concrete unity. A survey gathers information about one or more objects of interest. For example, in a hospital care medical drug usage survey, there can be three objects of interest: the hospital, the patient, and the drug. Objects of interest have two important roles in the survey domain. First, they segment questionnaire data by pointing to which real-world entity the data pertains. This segmentation is achieved through the relationship between *ObjectsOfInterest* and *Themes*. Second, objects of interest enable the ability to dynamically create entities about which questions are answered (Table 1). Considering the dynamic entity creation, the domain analysis pointed at two paths for entity creation: *themes* and *question items*. Theme-based entity creation happens when it is necessary to register a full account of the entities related to that object of interest, such as, for example, when it is necessary to build a patients list, with information such as name and age, before asking questions related to each patient drug usage while in hospital care. Question item-based entity creation happens when the answers to a question determine an entity creation. That would be the case when, in a question, a list of drugs (each a *QuestionItem*) is selected and used as input for making a set of questions about each drug and its application in patient care. This second scenario is the reason for the relationship between *QuestionItem* and *ObjectOfInterest*.

As previously mentioned, *NavigationItems*, whose concrete representations are *Theme*, *QuestionSet*, *Question*, and *QuestionItem*, hold the information necessary to model questionnaire navigation flow and state. As such, they must store expressions that define if each item should be visible or not, as well as a list of validations and triggers that allow to perform some level of answer quality control and data consistency guarantees. Model navigation items, validations, and triggers rely heavily on Boolean and arithmetic expressions to determine their behaviors.

Questionnaire metadata connects to the data model through 3 concepts: *Survey*, *ObjectOfInterest*, and *Measurement*, as shown in Fig. 3. Data and paradata structural models are closely related to a survey data collection operation (Fig. 1). As such, the relationship between concepts from metadata and concepts from data and paradata models is usually realized through conventions adopted in the data collection software. The usage of a dotted line indicates this lack of a concrete relationship between those concepts. A survey is basically a collection of observations made during an interview, which explains the relationship between *Survey* and *Interview*. Objects of interest are closely

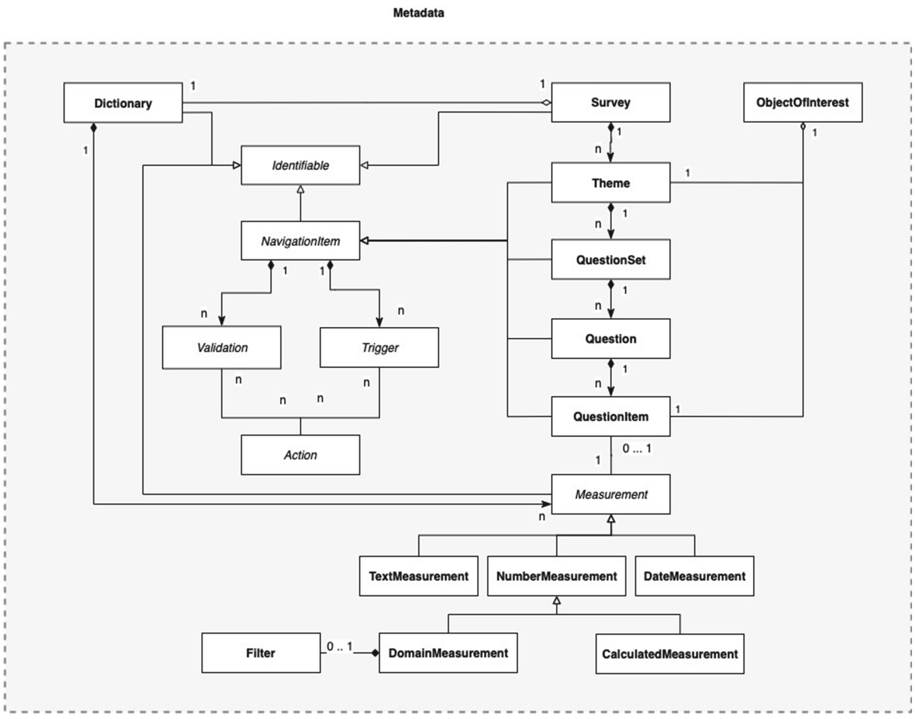

Fig. 2. Domain questionnaire metadata structural model.

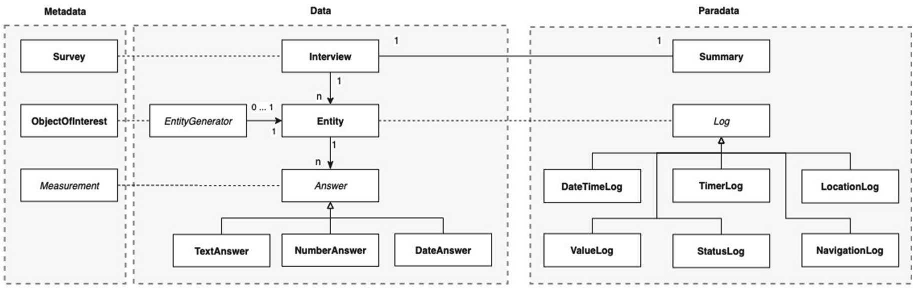

Fig. 3. Data and paradata structural models.

related to entity generators since an *EntityGenerator* defines rules for *Entity* creation. *Measurement* is the specification of an *Answer* format, type, meaning, and calculus rules.

Transversal Model Concepts. Expressions and rules are transversal concepts in the complex questionnaires domain model, frequently appearing as concept attributes. *Expressions* depend on primitives and operators. Operators mapped during domain analysis included boolean, arithmetic, comparison, set operators, and user-defined ones. Primitive Booleans, strings, and numbers coming from *Measurements* values or *Survey* constants are used in combination with operators in an *Expression* definition. *Rules* can be used to compute values in a rule-based *Action* or a *CalculatedMeasurement* and rely on operators and primitives, in the same fashion as expressions. Computation rules always evaluate to a number and use only arithmetic and aggregation operators.

Model Behavioral Aspects. The behavioral aspects of complex questionnaire models are closely related to questionnaire navigation flow and state of navigation items. Navigation flow and navigation items states are controlled by validations, triggers and expressions. Complex questionnaire behavioral model is closely related to *NavigationItems*. Figure 4 presents a full picture of the specialization structure for *QuestionItem*, providing a detailed picture of the attributes involved. Among the attributes depicted, *mandatory* and *visibility* play a key role in modeling questionnaire behavior. *Visibility* controls if a navigation item should be presented to the user. The evaluation of the expression associated with the *mandatory* attribute defines whether the question item should be answered and is used during validation activities together with other *NavigationItem validations* (run validations activity in Fig. 5).

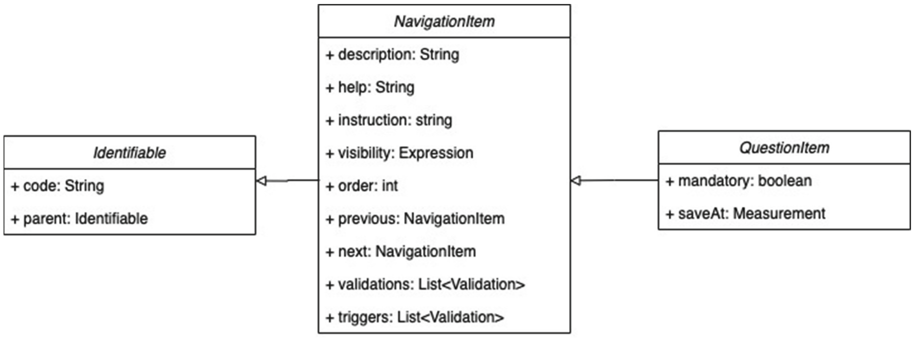

Fig. 4. *QuestionItem* specialization of *NavigationItem* [2].

Figure 5 presents the activities involved in a questionnaire navigation event. First, validations are run for the current *NavigationItem*. Otherwise, the event ends with the navigation aborted. Next, if measurements are consistent with validation rules, triggers are executed. If there is *NavigationItem* to be displayed after the current one, this item is retrieved. If not, the navigation is aborted. It is important to notice that navigation can happen up or down the questionnaire content hierarchy [2]. Finally, if a *NavigationItem* has been retrieved, its visibility is checked. If the item is visible, navigation is complete; if not, it is necessary to check again if there is a next *NavigationItem* to be presented.

Two additional aspects are important in the behavioral model. First, the questionnaire *NavigationItems* are organized in a hierarchy that affects navigation events resulting in horizontal and vertical navigation events. Horizontal navigation events happen when navigation occurs at the same hierarchy level. Vertical navigation events happen when there is a change in the hierarchy level. Second, the activities listed in Fig. 5 trigger transitions that affect the state of a *NavigationItem*. Mapped *NavigationItem* states are NOT ANSWERED, VIEWED, SKIPPED, and ANSWERED [2].

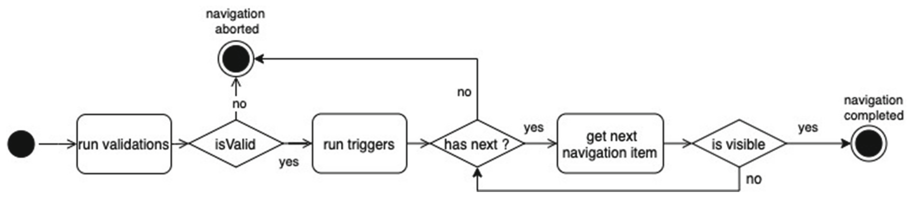

Fig. 5. Questionnaire navigation activities.

4 SLang Design

Research and industry practice bring little guidance on how to map domain models to DSLs concepts and syntax [18, 19]. This section describes the process of transforming the model described in Sect. 3 into a DSL for survey questionnaire specification, called SLang.

SLang was constructed to facilitate survey design and execution, focusing on questionnaire specification, and allowing to experiment with a model-driven approach and to probe its potential benefits. The focus on questionnaire specification was chosen due to its relevance to survey processes. SLang was designed by analyzing the model proposed in Sect. 3 and mapping its elements into potential language concepts. Next, potential language concepts were selected to be included and represented in SLang. This process is described in detail, starting with SLang's main design decisions in Sect. 4.2, followed by an overview of its abstract syntax and its concrete syntax in Sect. 4.3. Section 4.4 presents a survey questionnaire example, highlighting SLang's main features.

4.1 SLang Design Principles

Design guidelines for programming languages have been extensively discussed. Principles such as simplicity, security, fast translation, efficient object code, and readability are well-established programming language 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. Based on DSL design guidelines [20], the following principles guided SLang design.

Language Purpose. During domain analysis, questionnaire specification presented itself as the main challenge in the survey process, considering both the need to keep track of questionnaire design and the challenges in data collection systems development. When looking at the requirements involved in a survey process (Table 1), it becomes clear that there is a need for a formal communication tool that can tie different stages of the survey process helping avoid metadata scattering through the survey process, While improving communication among a multidisciplinary team that includes survey domain area experts, statistical and survey methodology experts, software engineers, IT infrastructure engineers, database administrators and business analytics. As such, SLang and its models were designed with the main purpose of survey questionnaire specification.

Since its purpose is complex questionnaire specification, SLang does not contemplate the questionnaire presentation aspects that directly affect the user interface, and general aspects related to data collection systems, such as security and data analysis.

Language Realization. Complex survey questionnaires present the challenge of expressing non-linear information flows, be it interactions, relations, or state changes that are better tackled by visual languages [21]. On the other hand, the textual format is more 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 metamodel later, when and if a real need is established [22]. Besides, as described in Sect. 2, no DSLs were found with the specific purpose of questionnaire specification, limiting the possibilities for language reuse. As such, a decision was made to create a textual language. For simplicity reasons, it was decided that SLang would initially be a non-executable language with pre-processor characteristics [8]. This decision does not preclude the possibility of making SLang executable in the future.

The usage of more general-purpose languages, such as UML or XML, were considered, but language workbenches had a series of advantages. First, they offer DSL users better editing experience and allow DSL designers to create custom editors with functionality similar to those of modern IDEs [10, 23]. Second, language workbenches can 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 [12]. Third, language workbenches bring the possibility of illustrative executions [24]. Finally, the available language workbenches offered the possibility of bringing SLang closer to IBGE social survey data collection infrastructure endpoints which is currently Android-based.

Language Content. Language content should reflect only the necessary domain concepts, keeping it simple, while avoiding unnecessary generality. The number of language elements should be limited by eliminating, whenever possible, conceptual redundancy and inefficient language elements [20]. With the stated language purpose in mind, the first decision made was to focus on metadata specification leaving data and paradata portions of the model presented in Sect. 3 untouched. The process of domain analysis and modeling eliminated most of the conceptual redundancy and unnecessary generalizations. Careful thought was also given to expressions. Questionnaire navigation is controlled by visibility attributes, usually expressed as Boolean expressions with *Measurement* values as operands. The level of complexity for those expressions is limitless, considering data collected during domain analysis. As such, the option was made to limit the operators available to those described in Sect. 4.3.

4.2 Abstract Syntax

An *abstract syntax tree* (AST) is a data structure that represents the abstract syntax used by compilers to represent and manipulate programs. Aspects of SLang AST are presented Fig. 6 and Fig. 7, in which rectangles represent nodes or concepts, arrows link parent and child concepts, forming the AST branches, and dotted arrows specify non-mandatory parent/child relationships.

The questionnaire hierarchy is presented in Fig. 6. The diagram was enriched to highlight two important aspects of SLang. *NavigationItems* generalization, represented as grayed out elements, and *ObjectOfInterest* generators, which are possibly either *Theme* or *QuestionItem.*

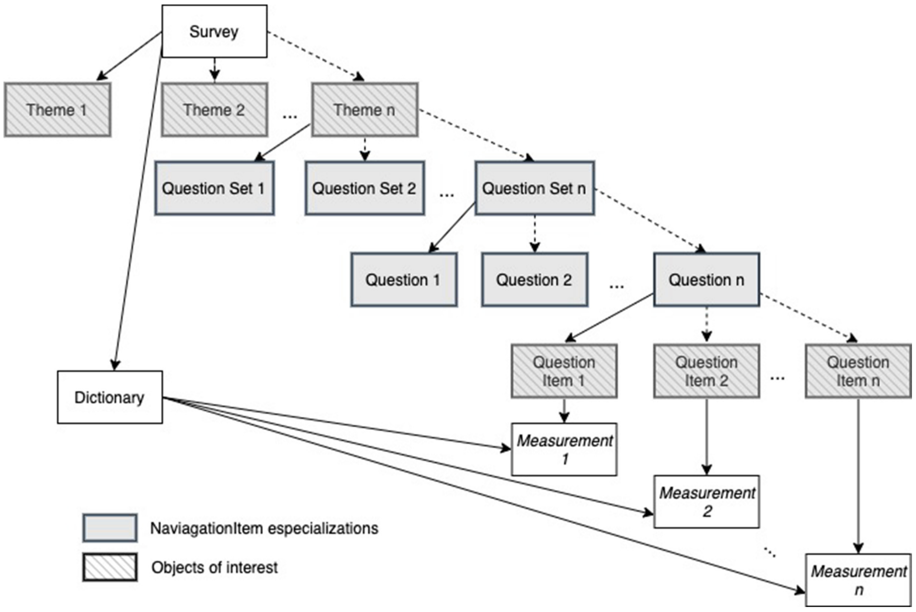

Fig. 6. AST from a questionnaire concept structure perspective.

The SLang root concept is *Survey*. *Dictionary* and *Survey* are related in a one-to-one composition, which is slightly different from the questionnaire model. Initially, SLang was designed to have *Dictionary* as a second root concept. The decision to model it as a child of *Survey* was taken after interviewing domain experts, who pointed out that dictionaries are survey-specific. There is limited opportunity for *Dictionary* model reuse and deriving it from *Survey* makes questionnaire modeling easier.

Figure 7 represents the navigation item ASTs. *NavigationItem* replicates the questionnaire model by providing the generic type that *Theme*, *QuestionSet*, *Question*, and *QuestionItem* specialize. Each *NavigationItem* can have multiple validations and triggers, which form branches of the AST. Each *Trigger* is connected to an *Action.* An *Action* affects one or more *Measurements.* The semantics of each one of those elements reflect the domain model. *NavigationItem* helps define the flow of answering a questionnaire. *Validation* is responsible for guiding how to provide adequate and valid answers. *Triggers* are responsible for guaranteeing data consistency among *Measurements.*

As presented in the model, *Expressions* are a transversal concept of SLang, providing means to represent intended questionnaire presentation flow and rule informed data consistency checks and actions. As such, most of the concepts are connected to one or more *Expression* concepts. SLang expressions follow the usual representation for

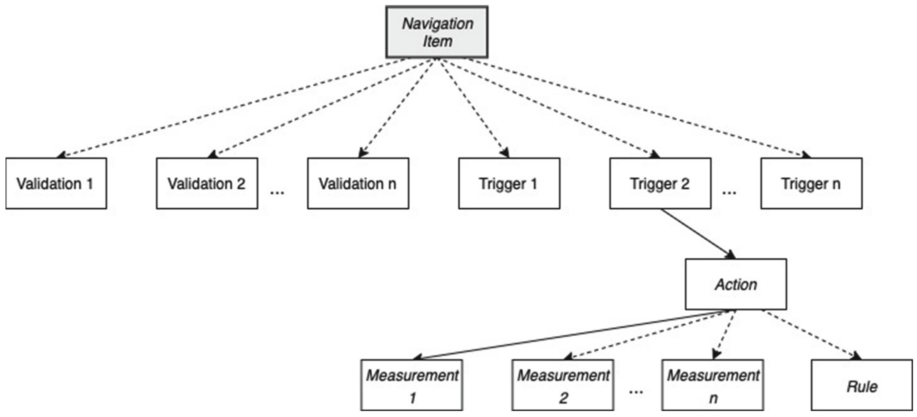

Fig. 7. AST from a *NavigationItem* perspective.

mathematical expressions as a hierarchy combining other expressions with operators and primitives [2, 8]. SLang supports three primitive types: number, Boolean, date and text. Primitive types serve as wrappers for measurements, constants and literal values. Operators wrap arithmetical, logic, and comparison operators. *Expressions*, *Operators*, and *Primitives* are used to check if expressions are valid by matching types.

4.3 Concrete Syntax

The *concrete syntax* of a language defines the notation with which users can express programs. In the case of SLang, a program is, actually, a questionnaire specification. The way questionnaires are expressed vary considerably, depending on the user background in terms of IT tools: some use spreadsheets, while others adopt text documents. Such a scenario makes SLang concrete syntax design decisions harder. In Slang, the concrete syntax is close to the textual writing of a questionnaire and uses sensible defaults, consistent choice of style, indentation, and conventions.

SLang concrete syntax is based on five main concepts: **Survey**, **Dictionary**, **Theme**, **QuestionSet**, **Question**, and **Item**. **Survey** is the root concept and, whenever created, a **Dictionary** is also created. **Theme**, **QuestionSet**, **Question**, and **Item** are nested elements from Survey and are progressively indented to represent their hierarchy. Each of those main elements has a list of attributes.

Attributes are the next level for SLang concrete syntax. For example, Table 2 shows **Theme** attributes. Only four attributes are mandatory: code, description, order, and object. The order attribute is grayed out because it is not defined by the questionnaire designer and is derived from the questionnaire hierarchical organization.

The *Identifiable* concept plays a key role in SLang concrete syntax. Its attributes, code and alias, are unique and play a key role in allowing the user to navigate the created model. *Identifiable* codes are automatically filled for all concept instances, except for *Survey*. An alias can be included whenever the questionnaire designer feels it is necessary for model clarity. *Survey*, *Theme*, and *Measurement* concepts require easy identification, as the user creates a questionnaire, and needs to review what was done. As such, either

Table 2. **Theme** attributes.

Theme Attribute	Mandatory	Description
code	x	string that holds entity unique identification
alias		String that holds a unique alias for the created theme
description	x	string for navigation item description
help		string explaining the navigation item
instruction		string for information regarding answering the navigation item
order	x	integer indicating the order of the navigation item within its level
visibility		*Expression* indicating whether the *NavigationItem* should be visible or not
triggers		list of *Triggers* for the navigation item
validations		list of *Validations* for the navigation item
questionSets		list of *QuestionSets* for the navigation item
creator		*Expression* that indicates if this theme should generate entities about which data will be collected
object	x	*ObjectOfInterest* to which the theme is associated.

code or alias values are presented always in green for those concepts, as can be seen in the Sect. 4.4 listings.

Another important aspect of the concrete syntax are the *Measurement* types. Given the importance of those types for questionnaire specification, the choice was made to give them special styling by using the orange color. SLang uses three types for those concepts: text, number, and domain. During domain analysis, modeling measurements was a challenge. For IT-related domain data pointed at multiple type possibilities many times in line with GPL primitive type system or relational database data types. Still, the complexity of those type systems was confusing for most survey domain experts. As such, when defining the abstract and the concrete syntaxes, the types remaining were those most easily absorbed by survey domain experts. To achieve the level of detailing additional attributes were included, with sensible default rules, to allow their mapping to the more complex software-related type systems. Table 3 presents the attributes of a *NumberMeasurement* with the associated default values.

Table 3. *NumberMeasurement* attributes.

NumberMeasurement Attributes	Default value	Description
code	-	*Measurement* unique identification
alias	-	*Measurement* unique alias for the created theme
type	-	number
size	3	integer representing the number of digits
decimal	0	Integer representing the number of decimal places

Expressions and *Rules* are determinant for specifying navigation flow [2, 8]. *Expressions* combine primitives and operators. Primitives might be literals or *Measurements*. As such, a Measurement code or alias can be used in expressions. *Rules* use the same paradigm. Table 4 presents a list of currently supported operators with concrete syntax examples.

4.4 SLang Code Snippets

When designing SLang the main idea was to allow the questionnaire designer to write its specification as text. Since this work started [2, 8], SLang syntax has already been adjusted to include some minor improvements to make it more friendly. Listing 1 presents how main concepts are expressed, including an example of the usage of **Survey**, **Theme**, **Question**, and **Item**. The language has been updated to allow suppressing the usage of **QuestionSet** for *Themes* with only one *QuestionSet*. The example also presents the usage of comments.

Table 4. SLang supported operators.

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

```
Survey: A demographic survey (S0001)
    description: A demonstration survey
    version: 1 !! allows controlling changes in the survey specification
    constants:
       min_wage: number 7.0
       reference_date: date 08/01/2019
    object: household  ( an address that serves as a dwelling for one family )
   Theme: Inhabitants information (T01)
      object: household
      instructions: Here go instructions for a potential interviewer
      help: Here goes an explanation of this theme
        Question: 1.1.1 - How many people lived in this household on {reference_date}?
           Item:
              saveAt: number qty_people_household
        Question: 1.1.2 - How many children with ages between zero and nine (include
        newborns) lived in this household on {reference_date}?
          Item:
              saveAt: number qty_children_household
```

Listing 1. Questionnaire content survey concepts.

Listing 2 presents an example of how a **Dictionary** is defined, including an example of a *CalculatedMeasurement*, which uses the *Rule* concept as an attribute of *Measurement*. Listing 3 presents the usage of *Validations* and *Triggers* in the context of specifying a question validation and measurement consistency through triggers.

```
Dictionary: A demographic survey dictionary (D0001)
    alias: household_proprietor
    code: V0003
    type: text
    size: 100
    alias: resident_income
    code:V0060
    type: number
    precision: 11
    scale: 2
    alias: household_income
    code: V0060t
    type: number
    precision: 14
    scale: 2
    rule: sum(resident_income)
```

Listing 2. SLang Dictionary example.

```
Question: 8.1.1 - How many sons and daughters born alive until {reference_date}?
    visibility: sex == 2 and age >= 10
    mandatory: false
    Item:
        saveAt: 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
        saveAt: number V0802
        Triggers:
            action: input(V0802 + V0803)
            measurements: qty_children_born_alive
         Validations:
            type: ERROR
            expression: qty_children_born_alive > 1 and qty_children_born_alive 30
            message: "The number of children born alive is invalid."
    Item: How many women?
        visibility: V0801 == 1
        saveAt: number V0803
        Triggers:
            action: input(V0802 + V0803)
            measurements: qty_children_born_alive
```

Listing 3. *Validations* and *Triggers* expressed in Slang.

5 SLang Implementation

SLang was prototyped using the MPS language workbench. This prototype provided the environment for experimenting and validating SLang as a model-driven approach, applied to the complex survey questionnaire domain. The validation process included two stages: real-world questionnaire specification using SLang and the usage of questionnaire model transformations applied in a software called SInterviewer used by real stakeholders to perform mocked interviews.

In the first stage, two real-world questionnaire specifications were encoded in SLang, using this prototype implementation. This part of the experiments validated the expressiveness of SLang. Then, mock-up surveys were run, using the encoded questionnaires, on top of a survey environment, called SInterviewer. This section describes some aspects of SLang prototype implementation, important aspects of SInterviewer, and the results of the prototype validation process.

5.1 SLang Implementations Highlights

The SLang prototype was built on top of JetBrains Meta Programming System (MPS). Five workbenches were first considered for the implementation of SLang: Spoofax, XText, Rascal, MetaEdit + and MPS. The decision to adopt MPS was influenced by the fact that only MPS had a projectional editor, that is, an editor that makes it possible to create, edit and interact with one or more ASTs, avoiding the need to use parser tools [8, 23]. Another aspect that influenced this decision was a broad Android-based data collection infrastructure used at IBGE and the fact that MPS provides Java compatibility, hence providing means for model transformations outputting Java code.

The process of prototyping SLang started with concepts mapping, which defined the AST creation rules and the SLang base structure. Then, the concrete syntax was enforced using the MPS editors. Finally, behavior and static semantics were added using the MPS Behavior, Constraint, and Type System aspects.

Behavior aspects made possible, for example, to attribute default values to questionnaire model properties, and to create and manipulate child nodes and references using MPS concept constructors and MPS concept methods. Static semantics was established through MPS Constraint Aspects and Type System Aspects. Constraint aspects provided, among other things, control of where concepts are allowed, validation for properties values, answer options control. Type system aspects were used for semantic aspects that could not be modeled using MPS base concepts, behavior aspects, and constraint aspects. For example, preventing nodes with the same name to exist in a specific scope could not be done using concept structure or constraint aspects. Constraint and type systems aspects provided hooks used by MPS to implement context assistance and error reporting, in the final language IDE generated using MPS [2].

5.2 SInterviewer

SInterviewer is a data-intensive mobile application, built on top of the Android platform, to collect questionnaire data and paradata. Its architecture is an evolution of the data collection software developed for IBGE Housing and Social Surveys, which included the 2017 Agricultural Census and the 2020 Demographic Census.

The SInterviewer engine is based on questionnaire metadata specified in Json format. As such, the application is configured during packaging for one survey at a time. That means it is impossible to run multiple surveys in the same Android application instance, that is, and each survey has its own app.

When the app is started, questionnaire Json metadata deserialization is triggered. The deserialized model is used through the services layer to control questionnaire navigation, persistence of answers and general application functionality, such as closing questionnaires and registering interview observations. Although being possible to configure the app differently, the SInterviewer presentation unit is a question, which means that the app user answers one question at a time. SInterviewer processes a questionnaire navigation request in a similar fashion to the behavioral model proposed in Fig. 5, but with additional activities for language persistence and the necessary checks to fully allow navigation item state control.

Two aspects are relevant for the usage of SInterviewer paired with SLang. First, SInterviewer mixes presentation aspects with questionnaire metadata. This poses a challenge that was mitigated by the usage of sensible defaults when implementing SLang to SInterview notation transformation. A second factor is that SInterviewer parses expressions for deciding if an answer is valid and if a question should be visible. The issue here is related to expression complexity. Not all aspects of SInterviewer are supported by SLang. As such, for certain behaviors possible in SInterviewer to be achieved from a SLang coded questionnaire, manual intervention is necessary for the outputs of the model transformation process. The option was made not to make this intervention. Still, this specificity should be addressed by a language evolution plan.

Data collected with SInterviewer can be extracted in two ways: local data extraction or synchronization with a backend. Local data extraction allows to export data in two formats: CSV or Json. If backend synchronization is enabled, data will periodically be sent to a backend and stored in relational databases.

5.3 SLang Usage in a Real-World Context

Two real-world questionnaire specifications were modeled using SLang and used in mock interviews using SInterviewer [2]. Table 5 presents some statistics about the modeled questionnaires providing some insights on their complexity. The number of *Themes* and *QuestionSets* were the same because both surveys had only one block of questions in each Theme.

Table 5. Modeled real-world survey questionnaire statistics [2].

Characteristic	Survey 1	Survey 2
Objects of interest	1	8
Themes	18	41
Question sets	18	41
Questions	102	185
Question items	102	893
Measurements	107	957
Validations	34	134
Triggers	28	102

Survey 1 was about K-12 students' health aspects and included themes such as students' socio-economical aspects, family context, eating habits, and physical activity, among others [25]. It included around 102 questions composed of single choice multiple-choice answers questions, which implies one *QuestionItem* per *Question*. Each *QuestionItem* was connected to a DomainMeasurement that defined the answer domain. Another aspect was the usage of derived measurements, which accounts for the fact that we have more measurements then question items. This survey navigation logic was simple and included mostly visibility expressions for skipping themes or questions and cross-validation checks among questions (questions with related answers).

Survey 2 was a preliminary version of the 2017 Brazilian Agricultural Census [26]. This survey had a more complex navigation structure with a larger number of question items due to the extensive usage of many items in one question (that happens when you have a multiple-choice question where the respondent can select more than one item). Survey 2 made extensive usage of the objects of interest creation support of SLang and SInterviewer. This allowed exercising theme-based object of interest dynamic creation and question item-based object of interest creation. In practice, the number presented in the table corresponds to the number of *QuestionItems* and *Measurements* from a questionnaire in SInterviewer. The number of coded *QuestionItems* corresponds to 305, and the number of coded *Measurements* amounts to 369.

5.4 Model Transformations and Mock Interviews

While working on SLang prototype model, transformations were experimented with the implementation of two model to text (M2T) transformations: one to generate questionnaire to SQL schema; and another to generate SInterviewer questionnaire to json metadata. Both transformations were implemented using the MPS TextGen aspect.

The questionnaire to SQL schema transformation aimed at creating a relational scheme to store collected data in backend databases. The mapping between SLang and SQL compliant code starts with a relational scheme for each survey and a table for each object of interest. In each table, one column was created to store each measurement value. Column creation code was derived from *Measurement* attributes according to each measurement type, with *Identifiable* code attribute being used to define column names. Each table has a column "id" marked as primary key. Currently, it is not possible to use compound primary keys or customize the primary key column name. Validations and triggers presented a challenge considering the complexity of expressions. Still, they are respectively mapped to "check" and "create trigger" SQL statements on a best effort fashion. SQL constraints were also included in the output. For example, whenever the *Measurement* associated *QuestionItem* was mandatory, the SQL "not null" constraint was added.

The SInterviewer questionnaire Json metadata transformation generates questionnaire specification in Json format, which can be submitted to SInterviewer (see Sect. 5.3), where it is parsed and used to feed SInterviewer questionnaire engine. Here, there was extensive usage of sensible defaults. As mentioned in Sect. 4, SLang does not contemplate presentation aspects, which are, to a small extent, covered in SInterviewer questionnaire metadata Json notation.

5.5 SLang and SInterviewer Validation

Starting from the surveys mentioned in Sect. 5.2, backend databases and SInterviewer apps were created and set up for performing mock interviews for each survey. Figure 8 presents the validation scenario.

Mock interviews focused on performing the main navigation flows of questionnaires to check how well the data collection system would work. Interviewers had previous

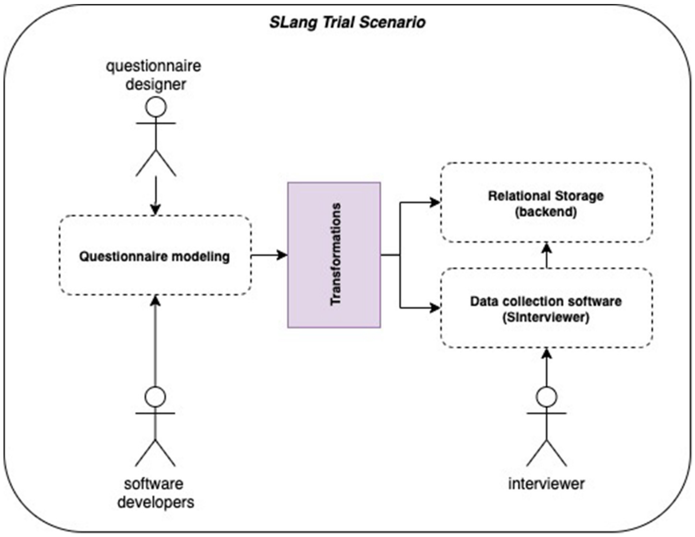

Fig. 8. SLang evaluation trial scenario.

experience with the original survey 1 and 2 data collection applications. Adjustments to questionnaire specification (required by interviewers) were simulated, to test the ability to quickly change questionnaire specifications and propagate them to the data collection system. In this scenario, the usage of the implemented transformations presented minor challenges specifically on *Expressions*, since excessively customized expressions do not have support both on SLang and SInterviewer. Also, the current version of SLang still lacks support for some operators and functions available on SInterviewer questionnaire metadata notation.

Overall, SInterviewer usage, with SLang generated input, performed well and received positive feedback from interviewers, when compared with previous data collection systems. Also, the time to deliver questionnaire changes was short, requiring only model adjustments, transformation execution and the publication of new versions of survey metadata. Some negative points were the impossibility of controlling presentation aspects on SInterviewer and the lack of support for highly customized expression rules.

The SQL generated schema also proved useful and avoided the need for domain expert data dictionaries that have to be translated into SQL schemas. It also standardized data constraints and integrity checks in schemas, which can potentially ease database schema administration for collected data relational databases. It also received positive feedback, since it eliminated the need to create and maintain data about the questionnaire design end, easing the integration between a mobile data collection and survey data collection management infrastructure.

The overall impression of stakeholders involved in the SLang mock trial was positive in the sense that having a questionnaire specification tool, and the possibility of developers working with model transformations generated a multitude of ideas that can be further explored in the future, such as: creating data tabulation systems input using model transformation, survey elements reuse through a survey models repository, survey comparisons, generation of inputs for legacy survey restoration, code generation to improve and ease systems integration, questionnaire version control support, among others.

6 Conclusions

This article presented the current status of SLang development, with improvements done to the domain model, and a broader scope evaluation of SLang in a mocked practical scenario. The effort to prototype and probe SLang as a model-driven approach to improve survey processes started with the goal of improving survey data collection systems through better communication between IT personnel and survey domain experts and through the improvement of software artifacts reuse, as well as elimination of unnecessary rework.

Questionnaire specification is the central point to achieving these goals. The usage of a model-driven approach proved a good strategy to maintain questionnaire specification centralized and to help generate artifacts that improved systems integration. Using the strategy of simulating a real survey scenario created an environment where stakeholders' feedback was rich in ideas for improvements, showing that there is considerable potential in the adoption of a model-driven approach.

Further work has been planned to further investigate and evaluate some points, before adopting SLang at an industrial scale. First, although SLang proved itself flexible enough for modeling complex questionnaires, usability evaluation is necessary to validate choices made in the concrete syntax and IDE approaches. Second, the evaluation stressed the potential benefits of combining a questionnaire presentation language with SLang. Third, survey IT infrastructure is diversified, and it is important to understand how far the usage of SLang can go. Further testing with code generation and transformations to integrate survey models with existing systems, such as survey metadata repository, survey data distribution, and publication systems, and data collection management systems, are necessary to better understand and evaluate SLang's full potential.

References

1. Saris, W.E., Galhofer, I.N.: Design, Evaluation, and Analysis of Questionaires for Survey Research, 2nd edn. Wiley, Hoboken (2014)
2. Araujo, L. Casanova, M.A., Leme, L., Furtado, A.: SLang: a domain-specific language for survey questionnaires. In: Proceedings of the 22nd International Conference on Enterprise Information Systems – Volume 2: ICEIS, pp. 133–144. Scitepress, Prague (2020)
3. Groves, R.M., et al.: Survey Methodology, 2nd edn. Wiley , Hoboken (2009)
4. GSIM and standards. https://statswiki.unece.org/display/gsim/GSIM+and+standards. Accessed 01 Aug 2020
5. DDI Alliance. https://www.ddialliance.org. Accessed 01 Aug 2020
6. SDMX. https://sdmx.org/. Accessed 01 Aug 2020
7. Borodin, A.V., Zavyalova, V.: Ontology-based semantic design of survey questionnaires. In: Proceeding of the 19th Conference of open Innovations Association (FRUCT), Jyvaskyla, pp. 10–15. IEEE (2016)

8. Araújo, L.C.: Model-driven questionnaires based on a domain specific language. Master dissertation presented to the Graduate Program in Informatics, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro (2019)
9. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
10. Nascimento, L.M.D., et al.: A systematic mapping study on domain-specific languages. In: The Seventh International Conference on Software Engineering Advances (ICSEA 2012), pp. 179–187. IARIA XPS Press, Lisboa (2012)
11. Kim, C.H., Grundy, J., Hosking, J.: A suit of visual languages for model-driven development of statistical surveys and services. J. Vis. Lang. Comput. **26**(99), 99–125 (2015)
12. Erdweg, S., et al.: Evaluating and comparing language workbenches: existing results and benchmarks for the future. Comput. Lang. Syst. Struct. **44**(A), 24–47 (2015)
13. Zhou, Y., Goto, Y., Cheng, J.: QSL: a specification language for e-questionnaire systems. In: IEEE 5th International Conference on Software Engineering and Service Science, Beijing, pp. 224–230. IEEE (2014)
14. Neighbors, J.M.: Software construction using components. Department of Information and Computer Science University of California, Irvine (1980)
15. Falbo, R.D.A., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engineering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), Ischia, pp. 351–358. ACM (2002)
16. Jatain, A., Goel, S.: Comparison of domain analysis methods in software reuse. Int. J. Inf. Technol. Knowl. Manag. **2**(2), 347–352 (2009)
17. Arango, G.: A brief introduction to domain analysis. In: SAC 1994 Proceedings of the 1994 ACM Symposium on Applied Computing, Phoenix, pp. 42–46. ACM (1994)
18. Czech, G., Moser, M., Pichler, J.: Best practices for domain-specific modeling. A systematic mapping study. In: 44th Euromicro Conference on Software Engineering and Advanced Applications, Prague, pp. 137–145. IEEE (2018)
19. Frank, U.: Some guidelines for the conception of domain-specific modeling languages. In: Proceedings of the 4th International Workshop on Enterprise Modelling and Information Systems Architectures, Hamburg, pp. 93–106 (2011)
20. Karsai, G., et al.: Design guidelines for domain specific languages. In: Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling, Florida (2009)
21. Wegeler, T., et al.: Evaluating the benefits of using domain-specific modeling languages - an experience report. In: Proceedings of the 2013 ACM Workshop on Domain-Specific Modeling, Indianapolis, pp. 7–12. ACM (2013)
22. Voelter, M., et al.: DSL Engineering. CreateSpace Independent Publishing Platform (2013)
23. Campagne, F.: The MPS Language Workbench, 3rd edn. Campagnelab (2016). http://books.campagnelab.org
24. Fowler, M.: A pedagogical framework for domain-specific languages. IEEE Softw. **26**(4), 13–14 (2009)
25. PeNSE. https://www.ibge.gov.br/en/statistics/social/justice-and-security/16837-national-survey-of-school-health-editions.html?=&t=o-que-e. Accessed 15 Jan 2020
26. IBGE Census of Agriculture 2017. https://www.ibge.gov.br/en/statistics/economic/agriculture-forestry-and-fishing/21929-2017-2017-censo-agropecuario-en.html?=&t=o-que-e. Accessed 18 Dec 2019