Model-centric and Phase-spanning Software Architecture for Surveys

Report on the Tool *Coast* and Lessons Learned

Thomas M. Prinz, Sebastian Apel, Raphael Bernhardt, Jan Plötner, and Anja Vetterlein
Course Evaluation Service and Chair of Software Technology
Friedrich Schiller University Jena
Jena, Germany
e-mail: {Thomas.Prinz, Sebastian.Apel, Raphael.Bernhardt, Jan.Ploetner, Anja.Vetterlein}@uni-jena.de

***Abstract*—Surveys are used in empirical sciences to answer research questions. Such surveys can be separated into five phases (e. g., planning and data collection), where each phase shares information with each other. Since the interdependencies between the phases are sometimes complex, it is helpful to have a software system, which supports each phase of a survey. There already exist such systems, which cover all of the phases. However, the implementations of the phases have usually strong limits; a more individual handling of the phases has to be done in external tools. But an external handling of the information would disrupt the links between the phases. The merging of the phases become a cumbersome task. This was one reason to build the new survey and report tool *Coast*. This paper presents a realization of the advanced requirements on this new survey system. The system keeps the links between the phases intact and is able to distribute surveys on different devices, e. g., paper and web. The focus on the meta-model of surveys makes that possible. The model is derived as a mathematical and as a data model. The data model builds a domain-specific language in order to construct the necessary parts of a survey. The architecture with its components and services is built around this language. Mainly, the architecture describes how the model of surveys is transferred and compiled. Its benefits and disadvantages provide lessons learned for other researchers and developers.**

Keywords–Survey; Architecture; Model; Coast; Tools.

I. Introduction

Surveys are prevalent in the empirical sciences (for example in psychology and sociology) [1]. In many cases, it is profitable to use surveys to reach a large and locally dispersed sample (e. g., [2]).

A survey can be separated in different phases: (1) Planning, design, and implementation, (2) data collection, (3) data preparation, (4) data analysis, and (5) reporting [3]. Each of the phases is interwoven with the others. For example, the reporting phase needs the specific questions asked in the survey as well as the different variables and scales from the implementation phase. In turn, the collected data can only be interpreted by knowing the scales and variables they belong to. So, it is beneficial to have a software system, which supports each of the phases of a survey and which is phase-spanning.

Such software tools are available on the market, e. g., *EvaSys* [4], *KwikSurveys* [5], *LimeSurvey* [6], *SurveyMonkey* [7], and *Unipark* [8]. Most of them cover all phases of a survey. They provide predefined or simple data analyses and reports. However, the data analysis and reporting phases are very individual processes and can become quite complex. For further analyses (e. g., multivariate analyses, machine learning) exceeding the standard repertoire, these tools offer the download of the raw data. This separation from the tool, unfortunately, disrupts the link between the data and the other phases of a survey — the individual data analysis in an external tool becomes a time consuming and cumbersome task.

This fact was one of the main reasons for the *Course Evaluation Center* at the Friedrich Schiller University Jena in the year 2009 to build its own survey and report tool, called *Coast*. It focuses on the integration of the analysis and report phases for advanced analyses. Although it is in successful use for the course evaluation of the university, the first version of *Coast* had its disadvantages. The disadvantages resulted from varying requirements, wrong design decisions, and surveys, which got more complicated than the architecture could handle. A monolithic architecture emerged, which made it hard and risky to implement changes in business logic. Although monolithic architectures are profitable in early-stage development [9], such architectures become more confusing during growth. That makes the architecture difficult to maintain and hard to understand [10][11].

As the questionnaires handled by *Coast* became more complex, the usage of the first version of *Coast* was not possible anymore. However, like a tool comparison in Section II-C will show, the currently available tools do not fulfil all our requirements. Usage of those tools cannot happen without significant changes on the questionnaires (and questions asked in the survey). Since we use longitudinal analyses [12] (which require equal questionnaires during each data collection), it is necessary to keep the existing structure intact. The overhead to change the questionnaires and to build workarounds to solve the requirements is high. Therefore, it is *not* possible without high efforts to use a different survey tool. This high effort made it necessary to transfer our tool *Coast* into a different architecture.

As a result, a new version of the *Coast* application is currently under development. This new version deals with the disadvantages of the first version by clarifying the architecture and modules as well as the model of ques-

tionnaires and reports. The architecture is build up around this model, which should offer a familiar development of questionnaires and reports. The model formulates a *Domain-Specific Language* (DSL) to handle the context-specific interaction between humans and machines. Our domain covers all phases of a survey and the model. In other words, one goal of *Coast* is to provide a DSL for surveys, which allows the construction of questionnaires and reports in a language that everyone understands who has knowledge in the domain.

This paper deals with our research question about the realization of a survey system that keeps links between the phases of surveys intact. We define requirements on such a system, especially those being not fulfilled by existing systems. A detailed presentation of the relationships and information of surveys and reports is done mathematically and as a data model. The data model builds a DSL for the construction of all the parts belonging to a survey. Based on this DSL, we examine the necessary components and services for the different phases of a survey and discuss them in terms of our software architecture. Benefits and disadvantages of our approach provide experiences for other researchers who have to realize similar research questions.

This paper is structured as follows: Subsequent to this introduction, the reader finds an overview about related work, our advanced requirements on a survey system, and a comparison of existing survey systems regarding those requirements (Section II). Section III describes the meta-model as a DSL, especially for questionnaires. This description is done (1) mathematically to provide an overview of the different information and relationships and (2) as a data model which gives more detailed information. Based on this data model, Section IV explains the architecture of our survey system with a high focus on the interactions between the phases of a survey. The benefits and problems of the proposed solution are shown afterwards in Section V. This paper concludes with a short outlook into future work in Section VI.

II. Related Work and Requirements

This section explains the five phases of surveys and defines our requirements on a survey system. An overview of existing solutions for creating, carrying out, and analysing surveys makes it further possible to compare the requirements for those solutions, subsequently.

A. Phases of an Electronic Survey

A survey can be separated into five phases [3]:

1) Planning, design, and implementation,
2) Data collection,
3) Data preparation,
4) Data analysis, and
5) Reporting.

The first phase (*planning, design, and implementation*) includes *planning* on the research questions you want to answer, *deciding* what survey questions are necessary to answer these research questions, and *specifying* who will be surveyed (the *population*). Furthermore, the survey will be implemented. As a result, there is a paper or web link that can be distributed to a sample of the population.

The population is surveyed with the implementation of the survey in the second phase. This phase is called *data collection* where the answers of the respondents for the questions are stored in a data set (for example in a database).

The third phase *data preparation* is necessary, because the collected data may contain malformed, incomplete, or wrong records. Sometimes, there are faults in the implementation or conception of the questionnaire, which have to be corrected. If other data sources are combined with the collected data, some data may have to be recoded. Summarized, the data has to be cleaned and prepared to be used in a *data analysis*, the fourth phase, afterwards.

In the analysis phase, the collected data are analysed in order to answer the research questions from the first phase. The results of the analysis, as well as general information about the survey, are finally summarized in detail in a report. The report is the outcome of the last phase, *reporting*.

B. Requirements

The primary requirement of a survey tool is its coverage of all mentioned phases of a survey. Besides this basic requirement, a tool in our context should fulfil the following functional requirements:

- DEV A questionnaire and report model should be usable for different **dev**ices (paper, web, smartphone, etc.).
- QAD New kinds of **q**uestion visualizations can be **ad**ded (sliders, timetables, etc.).
- AAD New kinds of **a**nalyses can be **ad**ded (group comparisons, regressions, etc.).
- MUL **Mul**tiple surveys should be analysed together in a single report.
- ADA Questionnaires and reports should allow **ada**ptivity, i. e., conditional branches.
- DIS The survey conduction should be **dis**tributed on other systems than the construction of surveys.
- PRI The survey conduction should be distributed also on **pri**vate systems.

Furthermore, it should fulfil the following two non-functional requirements:

- FLE The creation of questionnaires and reports should be **fle**xible.
- SCA All the phases should be **sca**lable.

C. Other Survey Tools and Their Comparison

Coast is not the only survey tool available on the market supporting the phases of a survey. There are well-established tools like *EvaSys* [4], *KwikSurveys* [5], *LimeSurvey* [6], *SurveyMonkey* [7], and *Unipark* [8]. All of them cover the previously introduced five phases of surveys. Almost all tools allow the creation of a questionnaire via drag and drop of the questions. The resulting questionnaire can be used for the data collection subsequently.

The tools are compared in Table I. The table compares licensing, type of operation, types of questionnaires, analysis tools, export of results, individualization of the layout, as well as extensibility of question types and data analysis algorithms. Concerning licensing, all tools offer a commercial license, which is usually accompanied by a managed

Table I. Survey tool comparison for license types, hosting, and devices for questionnaires, as well as export options for data and modifications to layout, question types, and analysis algorithms. The compared aspects were assigned to the requirements.

	Requirements	EvaSys	Unipark	LimeSurvey	KwikSurvey	SurveyMonkey
Commercial	-	✓	✓	✓	✓	✓
Open Source	-	✗	✗	✓	✗	✗
Free Version	-	(✓)	✗	✗	✓	✓
SaaS	DIS, PRI, SCA	✓	✓	✓	✓	✓
Self Hosting	DEV, PRI, SCA	✓	✗	✓	✗	✗
Web Questionnaire	DEV, PRI, DIS	✓	✓	✓	✓	✓
Paper Questionnaire	DEV	✓	✗	✗	✗	✗
Hybrid Questionnaire	DEV	✓	✗	✗	✗	✗
Analysis Tools	FLE	✓	✓	✓	✓	✓
HTML Export	FLE	✓	✓	✓	✓	✓
PDF Export	FLE	✓	✓	(✗)	✓	✓
Spreadsheet Export	FLE	✗	✓	✓	✓	✓
Raw Data Export	FLE	✓	✓	✓	✓	✓
Add. Question Types	QAD	✗	✗	✓	✗	✗
Add. Analysis Algorithms	AAD	✗	✗	✓	✗	✗
Individualization	FLE, ADA	✓	✓	✓	✗	✓
Multi-Survey-Report	MUL	✗	✗	(✓)	✗	✗

hosting and therefore a minimum of installation effort. Such a Software-as-a-Service (SaaS) offer is an advantage, especially with regard to the SCA, PRI, and DIS requirements. Some systems offer free test phases; others offer a permanent free use of the system with limited functionality. A hybrid concept (use of paper and web surveys) in relation to the DEV requirement can only be realized with the EvaSys tool. The realization of new question types and new analysis algorithms is only possible in the case of the open source application LimeSurvey through independent hosting and individual adaptation.

In conclusion, none of the tools considered would fully meet the intended requirements. In particular, the MUL requirement does not seem to be possible in any tool. The phases of data preparation, data analysis, and reporting are not separated in most of the tools and, therefore, merge smoothly. All the tools have the possibility to export the collected data for further research in external tools (e. g., *IBM SPSS* [13], *Excel* [14], or *R* [15]). Furthermore, they provide standardized reports, which include the questions of the questionnaires, frequencies, significance tests, and mean values, among other parameters. However, with all tools, the link between phases is lost in the case of individual analyses and the accompanying export of the collected data.

III. A Domain-Specific Language

The tools mentioned in the previous section have a focus on the development of the survey instead of a fine-granular and sophisticated data analysis and reporting. As mentioned before, a new focus on the data analysis and reporting was one reason to build an own survey tool *Coast*. Another main reason to build *Coast* was the missing coverage of our requirements of the existing survey tools.

As mentioned in Section II-A, conducting a survey is an almost well-defined process. The same holds true for the development of questionnaires (used to survey the population) and reports. It is our goal for *Coast* to offer a controlled and realistic modelling and development of them: We want to provide a DSL as a possible way to handle the interaction between humans and machines in a specific context. Our context contains all phases of a survey as well as how questionnaires, analyses, and reports are structured. In other words, the DSL can be used to model such questionnaires, analyses, and reports. In this paper, the DSL focuses on the description of the questionnaires to avoid repetition, since the description of the reports is quite similar.

Since the DSL should be used to define questionnaires in a language that everyone understands who has knowledge about questionnaires, the focus lies on *how* a questionnaire is structured and what kind of information is important. More concrete, the DSL describes the *meta-model* of the questionnaire. That means an instance of that meta-model is a description of one questionnaire. The instance of this description is one physical questionnaire, which can be answered by one respondent. The *meta-model* of questionnaires makes it possible to derive different kinds of surveys from the same model: For example, surveys represented online on PC, on paper, or on smartphones.

It is useful to inspect the structures of questionnaires from a mathematical perspective to get a general and compact model. Therefore, we considered questionnaire models and descriptions in the literature [16], interviewed psychologists in our department, and derived the following mathematical questionnaire *meta-model*.

A. Mathematically

One way to picture questionnaires is to consider them as computer programs: Questionnaires have a starting point and the respondent follows the questionnaire on a "path" question by question. Sometimes a path can branch out depending on previous answers and a respondent follows one or another. That means, the questionnaire forks — it allows *adaptivity*. In other places in the questionnaire, different paths join each other. It is like a program with branches. In other words, a questionnaire can be described as a (control flow) *graph* (definitions of graphs and paths can be found in [17, pp. 432] and [18, p. 1180]).

Before it is possible to define a questionnaire based on graphs, it has to be examined what the nodes of the graphs mean in this context. For this reason, we have to introduce some concepts of measurement theory.

1) Codomains and Variables: The theory of psychological measurements aims to study the properties of objects, especially of humans [19, p. 8]. In reality, the (psychological) properties, e. g., intelligence or motivation, are too complex and varied in such that the measurement has to coarsen and discretise them. Therefore, each measurement describes the mapping from the *true* domain to a set of discrete values [20, p. 22]. This set of discrete values is called the *codomain*.

Stevens describes four different kinds of codomains named *levels of measurement* [21]. Depending on the level of measurement of a codomain, it can be decided, which statistical analyses can be applied. The levels are *nominal*, *ordinal*, *interval*, and *ratio*. Starting from the *ordinal* level, the codomains are called *scales*. More information about the different levels can be found in [21].

It is necessary to decide carefully which level of measurement is suited for a specific measurement since it is difficult or impossible to transform the values of a low-level to a higher level measurement.

A property of an object obj is measured with the help of the values of a codomain $\mathcal{D}$. A measure assigns a value $val \in \mathcal{D}$ to obj and can be described as a pair (obj, val). If the same property is measured for the set of all objects $\mathcal{O}$ in a sample, it results in a mapping from those objects to the codomain $\mathcal{D}$. This mapping is called a *variable* [20, p. 22]:

Definition 1 (Variable): A variable V is a left total mapping from a sample set of all objects $\mathcal{O}$ to a codomain $\mathcal{D}$, $V\colon \mathcal{O} \mapsto \mathcal{D}$. The codomain of V is described with $\mathcal{D}(V)$.

A variable describes the characteristics of *one* property for different objects [22, p. 19]. However, in questionnaires, many of such properties should be measured. Therefore, there is a variable for each of the properties to measure.

2) Items, Pages, and Questionnaires: The measurement in questionnaires is done by asking the object (the *respondent*) with a set of instructions and questions — *items* [22, p. 19][23]. An item is a concrete question or request, which measures one or more variables.

Definition 2 (Item): An *item* I consists of an *instruction* and a set of enquired *variables*.

In almost all questionnaires, more than one item is presented to the respondent at the same time. They are grouped thematically on *pages*:

Definition 3 (Page): A *page* is a finite set of items.

Not each page has to appear for each respondent. For example, if a respondent has never done a job, a page with items about jobs is not suitable. In questionnaires, the exclusion of pages and items depends on previous answers of the respondent. There could be a previous item with the question "Do you ever had a job?" as an example whose answer includes or excludes the page about jobs. Questionnaires with conditional pages are called being *adaptive*. In adaptive questionnaires, *conditions* are allowing the control of the path a respondent follows:

Definition 4 (Condition): A *condition* on the variables $V_1, \ldots, V_m$, $m \geq 1$, is a left-total mapping from the Cartesian product of the codomains of the variables to the boolean set $\{true, false\}$:

$$\mathcal{D}(V_0) \times \cdots \times \mathcal{D}(V_m) \mapsto \{true, false\}$$

It is known from research that the structure of a questionnaire can influence the measurement results [24, S. 68 ff.]. The structure of a questionnaire seems to be important. For this reason, a questionnaire should not only be defined by an unordered set of pages and conditions. There has to be an order of the pages and items [25]. Sometimes, this order is described as the *item flow* and *sequence of questions* [26]. As mentioned before, it is promising to describe the structure of a questionnaire as a control flow graph, in our case as an acyclic, connected digraph [27, S. 547] (also proposed by Bethlehem [16]).

Definition 5 (Questionnaire): A *questionnaire* Q is a triple $(\mathbb{P}, \mathbb{E}, Cond)$. It consists of an acyclic, connected digraph $(\mathbb{P}, \mathbb{E})$ with a set of pages $\mathbb{P}$ and a set of *edges* $\mathbb{E}$; and a left-total mapping, $Cond$, which assigns a condition to each edge.

3) Measurement Methods and Surveys: A questionnaire can be used as a *measurement method*. A measurement method is applied for a single obj and a set of variables. For each of these variables var a pair (obj, var) is determined, $val \in \mathcal{D}(var)$. Assume the variables in a specific order within the measurement method. Then it is a mapping from the set of all objects of the sample to the Cartesian product of all codomains collected in the measurement method.

Definition 6 (Measurement Method): A *measurement method* $\mathcal{M}$ is a mapping from the set of objects $\mathcal{O}$ to the Cartesian product of all codomains of all measured variables $\mathcal{V}$:

$$\mathcal{M}\colon \mathcal{O} \mapsto \prod_{v \in \mathcal{V}} \mathcal{D}(v)$$

Remark 1: A questionnaire is a specific measurement method. The set of objects is the set of respondents. The set of all measured variables contains the variables in the questionnaire. Therefore, a questionnaire is also a mapping from an $object$ (respondent) to a set of values (the answers).

Since a questionnaire is a measurement method, a *survey* is a special *elicitation* in our context using a questionnaire [22, p. 18]. Elicitation is the application of a measurement method to a set of objects.

Definition 7 (Survey): A *survey* $\mathcal{S}$ contains a questionnaire Q and a set of *respondents* $\mathcal{O}$, $\mathcal{S} = (Q, \mathcal{O})$. The union of all measurements of all objects builds the results $\mathcal{R}$ of the survey $\mathcal{S}$:

$$\mathcal{R}(\mathcal{S}) = \bigcup_{o \in \mathcal{O}} Q(o)$$

B. Data Model

There are some mathematical structures, which are not trivial to implement, e. g., conditions. Some other structures are too unspecific yet to be used in software. An example is a codomain, which is an arbitrary accurate set in a mathematical sense. However, such a set cannot be implemented for discrete computers. For these reasons, the mathematical model has to be made concrete as a data model.

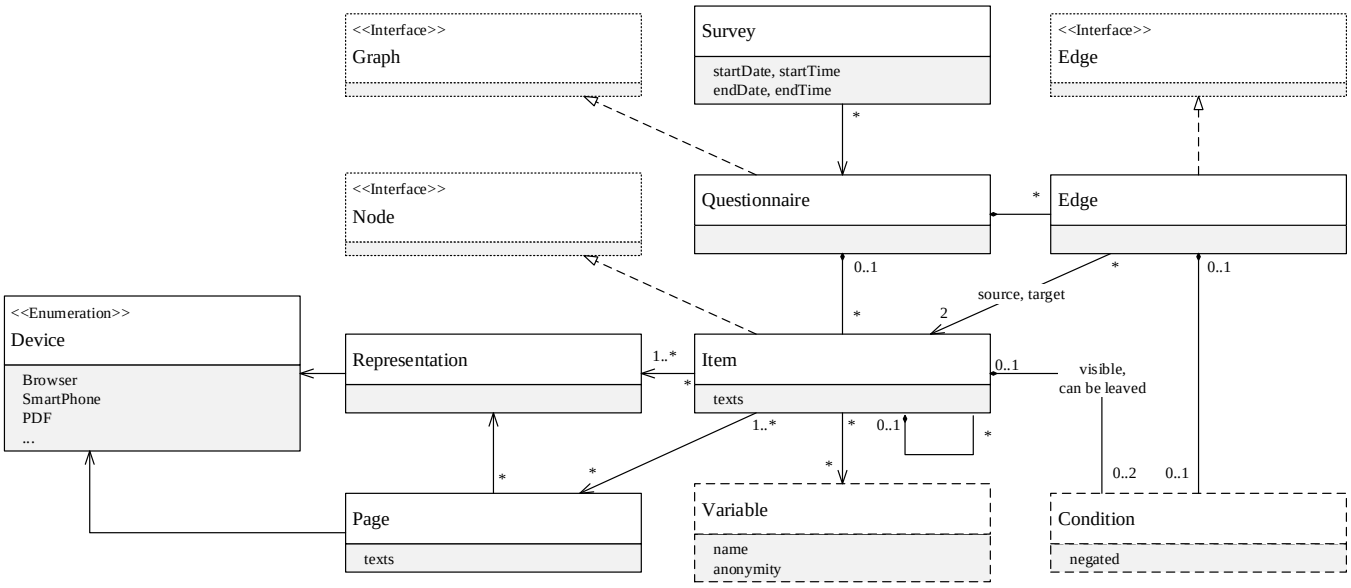

Figure 1. The questionnaire classes.

1) Questionnaires, Items, and Pages: Figure 1 shows the classes belonging to questionnaires and surveys. Classes with dashed lines have at least one association with a class in the current diagram but will be explained later.

The class *Survey* represents an elicitation using a questionnaire. Besides the questionnaire, a survey has a start and end date as well as a start and end time in the data model. The respondents (see Definition 7) are not illustrated in the class diagram since they are not of interest in this context of the data model and architecture yet.

Regarding Definition 5, a questionnaire is a graph consisting of pages and edges, where each edge has an assigned condition. A questionnaire is represented by the class *Questionnaire* in the class diagram. It implements an interface *Graph* giving it rudiment graph functionalities like getting the nodes and edges. That makes it also possible to apply general graph algorithms to questionnaires, e. g., topological ordering, data-flow analyses, etc.

Instead of referring to pages, the questionnaire has *Item*s as nodes in our data model. That infringes the mathematical formulations for reasons of generalization: Each questionnaire should contain the same items independently from the device, e. g., a browser, a smartphone application, or a PDF. The number of items, which should be displayed on the same page, varies for different devices because of display sizes. Therefore, the pages should be defined dependent on the device. For example, a smartphone display may be too small to show more than two items on a single page. A desktop device, however, enables at least five items per page. The items are the same for both devices, but the pages differ.

For these reasons, the questionnaire in the data model refers to items, where each item refers to a set of pages. Each item can only have up to one page per device as otherwise, it would be uncertain, which page should be used. A *Page* has a link to the *Device* enumeration in the class diagram. Since the page depends on the device, the items can be grouped differently on pages for diverging devices.

The *Item* in Figure 1 has a reflexive aggregation. Since items can become very complex, such complex items can be decomposed into *sub*items in the model, i. e., complex structures can be reduced to more simple ones. Although this is not explained in the mathematical model, this design decision supports reusing of structures. A battery of rating items (a set of questions with usually five or seven different, distinct answer possibilities) is a prominent example of such a complex item (cf. Figure 2). In our model, a rating battery with five different questions would be separated into an item representing the whole rating battery (in the example of Figure 2 the box with the header “Some questions”) and five subitems representing the five different questions.

As it can be seen in this example, the master item should have a different visualization than the subitems — it should group the subitems in a table-like form, where each row represents one subitem with the actual question. Although the visualization of the items is not part of the mathematical questionnaire model, there is a need for it in practice. For this reason, an item has at least one representation (the association to the class *Representation* in the diagram of Figure 1). Similar to a page, a representation is defined for a specific device since an item could be represented differently for example on a web or a paper survey. There is a restriction that each item has up to one representation for a specific device. Like the diagram shows, a page has different representations too since it must also be visualized.

Items and pages contain *texts*, which define, e. g., questions, instructions, and headers, corresponding to the mathematical model (Definition 2). The different texts should be stored language-separated such that a questionnaire can be used in different languages.

As a further extension to the mathematical model, each

Statements						
(*1=strongly disagree ... 3=neutral ... 5=strongly agree*)	*1*	*2*	*3*	*4*	*5*	*no answer*
1 The instructor creates a stimulating work atmosphere.	○	○	○	○	○	○
2 The course stimulated my interest in this topic.	○	○	○	○	○	○
3 The subject matters fit my level of knowledge.	○	○	○	○	○	○
4 Most of the participants attend the course regularly.	○	○	○	○	○	○
5 Overall, I am satisfied with the general conditions of this course.	○	○	○	○	○	○

Figure 2. Example of a battery of rating items.

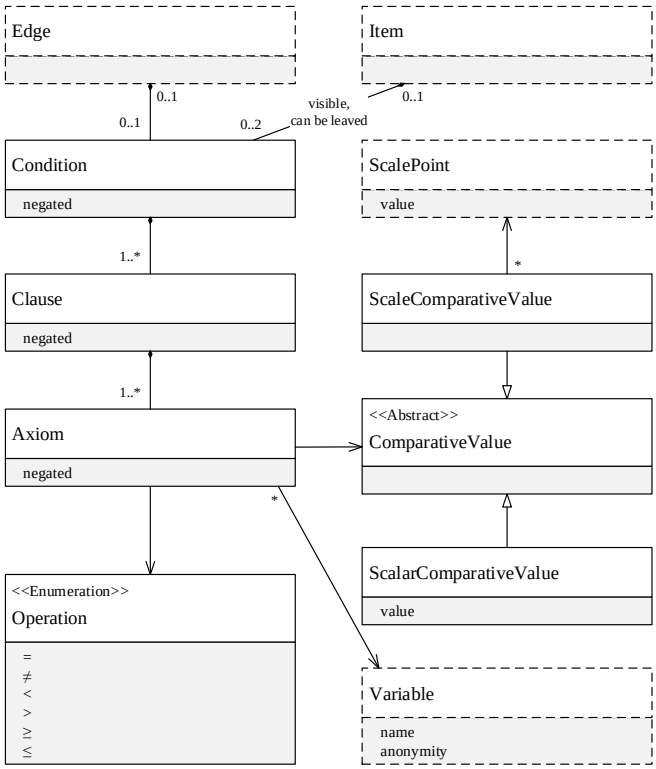

Figure 3. The condition classes.

item may have two conditions: (1) A condition whether the item is *visible* to the participant, and (2) a condition whether the participant *can leave* the item. The latter condition is practically necessary to avoid the reaching of the next page by a participant if some questions are required to answer. The former condition is an extension to the adaptivity of the questionnaire: Items can be visible on a page to specific groups of participants only.

Conditions are also used by the edges of the questionnaire following the mathematical model. In the class diagram of Figure 1, the questionnaire consists of edges beside the items. The class *Edge* implements an interface *Edge* and has a *source* item, where the edge starts, and a *target* item, where the edge ends. As a restriction, an edge cannot start or end in a subitem.

2) Conditions: As mentioned before, adaptivity can be realized by multiple outgoing edges of an item. This adaptivity is based on conditions following the mathematical model. If the condition holds true for an edge, then this edge of the questionnaire is subsequently followed. Mathematically, such a condition is easy to describe. However, the conditions in the data model should be interpretable or executable for a computer program. Therefore, it has to be described as a term (string) or as a parse-tree [28, pp. 45] in the data model. We decided to describe a condition as a parse-tree in disjunctive normal form (DNF). The advantages of using the DNF are that (1) each logical formula can be described in DNF and that (2) the resulting tree of clauses and axioms (the parse tree) has always the depth of 3. The latter makes it possible (a) to generate a simple UI for adding and describing new conditions, (b) to avoid parsing of terms in string format, and (c) to transform it to source code very fast. The disadvantages are obvious, (i) the possible bigger size of such conditions and (ii) the expected knowledge about logical equations in DNF and its transformations. However, the advantages prevail the disadvantages.

The class *Condition* in the class diagram of Figure 3 represents a condition in DNF. As an addition to the DNF, a condition can be *negated*. Each condition consists of at least one *Clause*, where each clause can be *negated* again and all clauses are connected by a logical OR ($clause_1 \vee \ldots \vee clause_n$, $n \geq 1$). A clause object connects different *Axiom*s with a logical AND, $axiom_1 \wedge \ldots \wedge axiom_n$, $n \geq 1$.

In our model, each axiom is a simple expression $(variable\ \ operator\ \ value)$. $variable$ is a reference to a variable (cf. class *Variable*), $operator$ is a relational operator, i. e., $=, \neq, <, >, \geq, \leq$, realized by the enumeration *Operation*. Eventually, $value$ is a comparative value. Since the comparative values can have different data types, e. g., a string, an integer, or a reference, they are realized as abstract class *ComparativeValue*. In the class diagram, two specializations are illustrated: A *ScalarComparativeValue*, which represents an integer, real, or string value, and a *ScaleComparativeValue*, which is associated with a specific scale point. Scale points are explained later. It is possible to add other specializations of the *ComparativeValue* class, e. g., one for handling complex macros.

3) Variables: The classes *Axiom* and *Item* are associated with the class *Variable* of the class diagram in Figure 4. It represents a variable of the mathematical model (Defini-

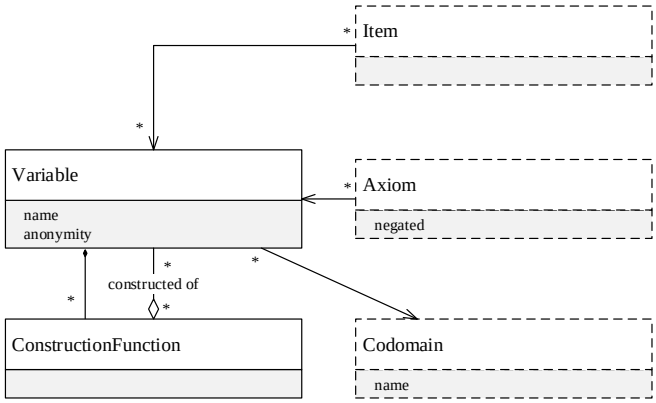

Figure 4. The classes of variables.

tion 1) and has a unique name and a rank of anonymity. The latter is technically important due for reasons of data security and anonymous surveys. For example, if there is a variable representing email addresses, the addresses should not be stored in the same context as other anonymous variables. Otherwise, the variables would lose their anonymity.

Another addition to the mathematical model is the possibility to define construction functions for variables. A construction function is a formula (e. g., source code), which describes the deriving of values for the variable based on the values of other variables. Each variable may have different *ConstructionFunction*s. A construction function again uses variables for their constructions. Obviously, each cyclic construction of variables should be avoided, i. e., no variable should be able to be constructed of itself.

4) Codomains, Scales, and Scale Points: Each variable uses an object of class *Codomain* following Definition 1 of the mathematical model. The classes belonging to codomains are shown in Figure 5.

For codomains, several specializations represent different levels of measurement as explained before and introduced by Stevens [21]. A *NominalCodomain* is a codomain having a relation (class *Relation*) defining equivalence between two values of this codomain. An *OrdinalScale* is a nominal codomain with an additional relation, which defines a linear order on the values of this domain. The class *IntervalScale* specializes the ordinal scale with two additional relations for plus and minus operations. At last, the class *RatioScale* is an interval scale with an identity element and additional multiplication and division relations.

Based on the measurement level one can decide which analyses are possible and could be applied on the collected data. For those analyses, it is good to know all the values in the codomain. For a large portion of codomains, there is a finite and enumerable set of discrete values describing it. These values are called *scale points* in the data model.

An object of the class *ScalePoint* consists of a value from the base set of the codomain (which will be explained later). Furthermore, a scale point has texts describing the textual answer printed to the respondent. For example, in the previously shown battery of rating items in Figure 2, the scale points of each statement are “strongly disagree”, “disagree”, “neutral”, “agree”, and “strongly agree” with the values 1 to 5. Most codomains have a list of such scale points. Furthermore, they have another list of *special* scale points. Special scale points are, for example, “default” and “no answer” scale points. As an example, each statement of Figure 2 has the value of the “default” scale point if the respondent saw it but has not answered; and it has the value of the “no answer” scale point if the respondent chooses the “no answer” checkbox.

5) Base Sets: As mentioned before, the values of the scale points have to be part of the base set of the codomain. In mathematics, it is easy to describe a set D with $D \subseteq \mathbb{R}$ as part of the real numbers or with $D = [5, 10]$ as all real numbers between 5 and 10 inclusively 5 and 10. In a data model, this has to be a part of the model too.

For this reason, the mathematical model was extended, and a *BaseSet* describes the set of values a codomain is based on. It is shown in the class diagram of Figure 6.

A base set describes typical number systems in the first place, for example, integer and real numbers. If the codomain contains texts, the base set is a set of strings (words), i. e., the Kleene star on the set of all characters. Alternatively, a base set can be a set of arbitrary objects, e. g., pictures or persons.

As base sets can look different, the class *BaseSet* is abstract. Therefore, a codomain must use one of its specializations. The model defines four concrete specializations: (1) A *CharacterSet* defining arbitrary texts, (2) an *ObjectSet* for arbitrary objects, (3) *Integer* for integer values, and (4) *Real* for real values. The latter both classes *Integer* and *Real* are specializations of the class *NumberSet*. Each number set represents a number systems and may define an interval between *from* and *to*. The interval has a specific distance between all the values (*stepsize*). The *NumberSet* is abstract with only two specializations *Integer* and *Real*.

With the description of the base sets, the mathematical model was transferred to a data model. As the reader can check, the data model differs from the mathematical model mostly for reasons of generalization in the resulting architecture and for more details and simplifications of mathematical terms.

6) Reports: The report model is similar to the questionnaire model, however, the model will not be introduced in detail in the context of this paper. Instead, the following explains the report model in short.

The class *Report* is a graph like a questionnaire. It consists of different *Parts* (pendants of items in questionnaires) and *ReportEdge*s. The edges of a report use conditions again. This allows the creation of individual reports based on a single report model. Report edges connect the parts in the report. Each part has different *ComputationalRepresentation*s, up to one for each *OutputDevice*. There could be different output devices, e. g., for online or paper reports. The representations contain information about the visualization of a part and define *Calculation*s. A calculation illustrates a function performed on the collected data of the part. For doing this, it has different *Parameter*s specifying the input and output information of the function. The parameters could

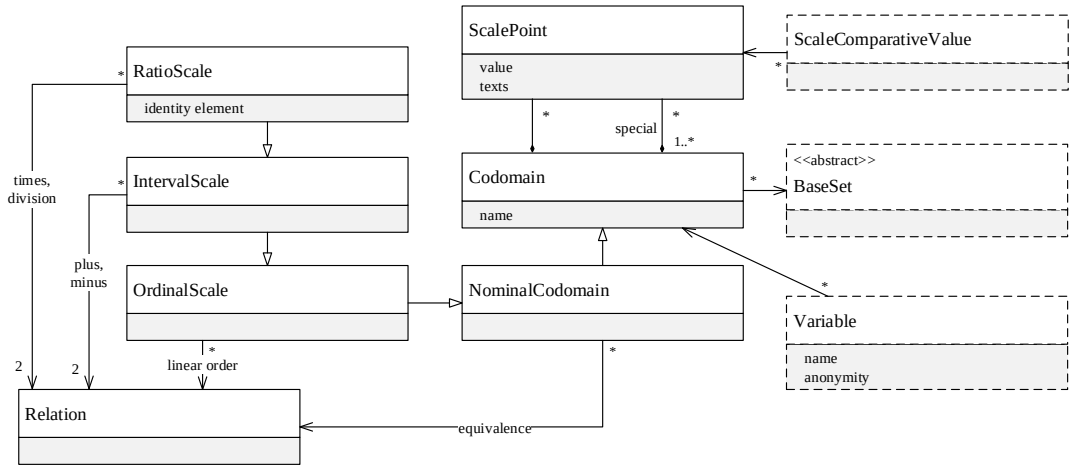

Figure 5. The codomain classes.

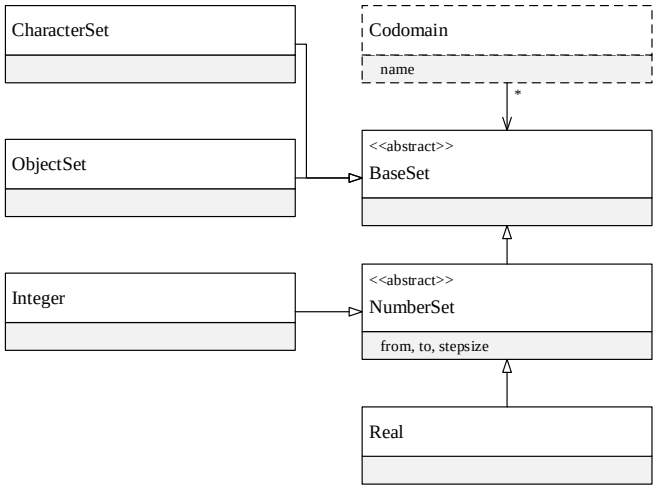

Figure 6. The base set classes.

be scalars or variables.

Besides the *Report* class, there are *EvaluationProject*s configuring the report. The evaluation projects define *DataSource*s, which specify the places of different data collections. This makes it possible to combine different data sources (also external data) in a single report. Evaluation projects also define sets of *Filter*s. Filters filtrate the data sources and create groups. For example, in a longitudinal survey, one could filtrate the data based on the year and create a group and data for each year. This allows to compare groups in the reports. Furthermore, each collection of filters ends in a single report.

IV. A System Architecture for Surveys

Our central architectural concept is to build the architecture around the questionnaire and report models. These models form the language for psychologists, sociologists, and researchers of other empirical disciplines to describe their needs. That means, the models also have to contain the processes and logics, which the researcher needs; *or* the architecture has to contain these processes and logics. Altogether, this is the *domain* of a survey: the domain covers all phases of conducting a survey and the models, processes, and logics. Since the models are part of the architecture, the architecture does finally represent the complete domain. In the topic of software engineering, this is a *Domain-driven Design* (DDD) approach [29].

The questionnaire model was derived to be a DSL to describe a questionnaire as near as possible to the daily experiences of researchers in empirical sciences (the users). For this reason, researchers can describe their questionnaires on their own without deep experiences of the system. That means, the questionnaires are developed by the domain experts and not by the developers of the system like originally done in classic DDD approaches. In other words, the users create formal models of questionnaires, which are afterwards able to be used to conduct web or paper surveys and to perform analyses and create reports. What the system has to achieve is to handle such formal models and to give an infrastructure for transformations, compilations, interpretations, executions, and other derivations.

The focus on a formal domain-specific model and its transformations into runnable code or reports as well as the focus on the infrastructure for such models make it a *model-driven engineering* (MDE) approach. MDE is the transition of DDD to software architectural decisions. In classic MDE a software is automatically derived from a formal model similar to the way our questionnaires are compiled automatically to web surveys [11]. Prominent examples of MDE are the *model-driven architecture* of the *Object Management Group* [30] and the *Eclipse Modeling Framework* [31]. Both focus on the *Unified Modeling Language* (UML) and provide an infrastructure of transformations from UML into the source code. In general, however, MDE is independent of UML and can be applied to arbitrary DSLs.

Our architecture describes the infrastructure and the transformations of the questionnaire and report model during the five phases of conducting a survey. Most parts of the

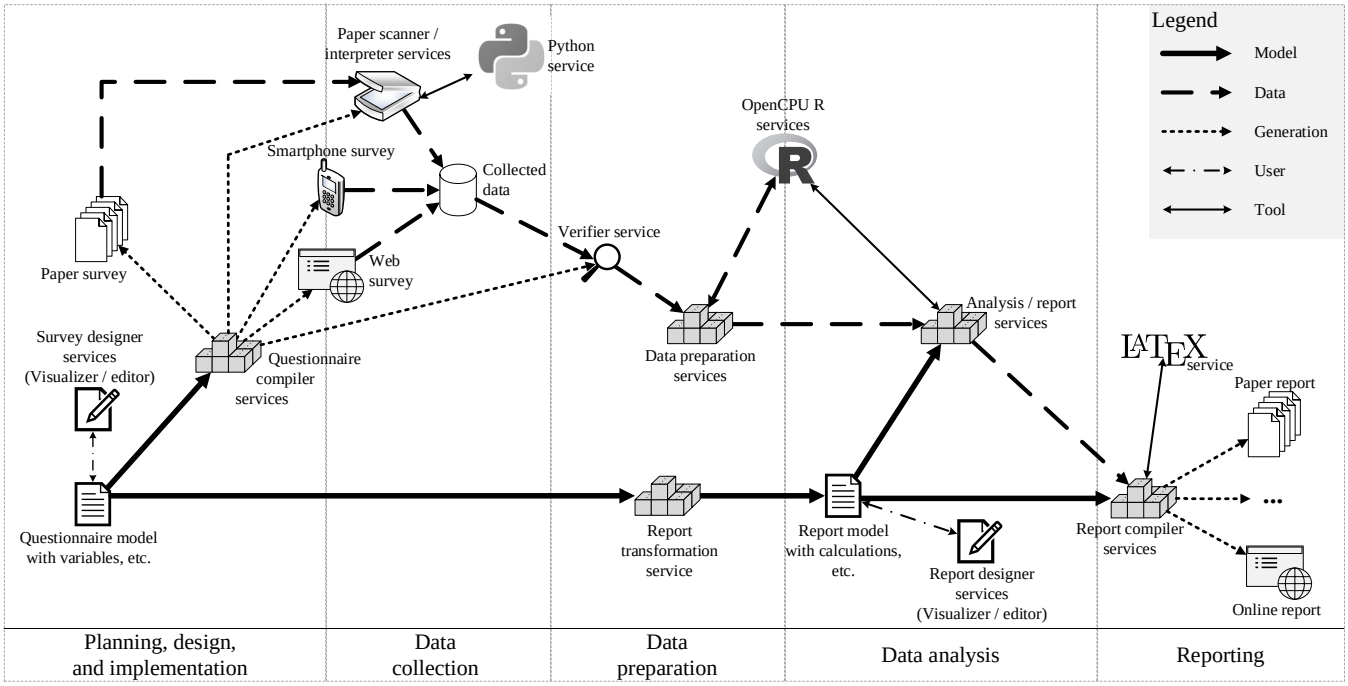

Figure 7. The architecture of the *Coast* system regarding the survey phases.

architecture use a questionnaire or report as input. For example, a questionnaire designer is a part of the architecture and receives a (possibly empty) questionnaire as input and produces a modified questionnaire as output.

Figure 7 shows our simplified architecture (infrastructure) separated into the five phases of (1) planning, design, and implementation, (2) data collection, (3) data preparation, (4) data analysis, and (5) reporting. The architecture is explained in the following.

A. Information Flows in the Architecture

The diagram contains different edges describing different flows between the phases. The most important flow is the bold one: The *model* flow. The model flow describes a questionnaire or report described in the DSL. It can be interpreted as a specification, which is sent to some algorithms or services (the term "service" will be explained later).

From the perspective of the user (the empirical researcher), the *data* flow seems to be the most important. It describes the collected data of the surveys and how it is transferred. The data flows are illustrated as dashed bold edges.

Regarding the MDE approach, the dotted edges are of interest. They show automatically generated programs, documents, etc. The primary input for those automatically generated elements are the user-generated models. Although there is more interaction between the user and the infrastructure as illustrated in Figure 7, this interaction is reduced to the UI tools named *survey* and *report designer* for reasons of clarity. The interactions between the user and the services and models are illustrated as small dashed and dotted lines. Eventually, interactions between external tools and the architecture are illustrated as small solid edges.

B. Services in the Architecture

Since the architecture covers the phases of conducting a survey, there is a natural flow from the left side of Figure 7 to the right side. The following description of the different parts of the architecture follows this flow starting in the first phase: planning, design, and implementation.

It all begins in the *Survey designer services*. These allow the user to model a questionnaire using our questionnaire DSL. The services provide a visual designer and editor for changing texts, variables, and others. They handle the creation and modification of necessary instances of the classes of the questionnaire meta-model. As a result of the designer, there is a well-specified *questionnaire model*.

If the user decides that the questionnaire is finished and should be used in a paper and web survey, for example, the model of the questionnaire is given to the *questionnaire compiler services* via the model flow. The compiler services implement the questionnaire and translate the model into a paper and web survey (illustrated as generation flows). The web survey can be accessed from the web and the results are stored directly in a data collection. This is illustrated as outgoing data flow from the web survey to the *collected data*.

In the case of a paper survey, a printable survey will be generated by the compilers. The printed questionnaires can be given to the participants. Since the survey results are not available digital, the compilers produce templates of the questionnaire based on the printable survey too. These templates can be used for scanning the printed and completed

papers in the *paper scanner* and *interpreter services*. During the scanning process, the data is extracted and the results are stored; illustrated as data flow from the scanner services to the *collected data*. The scanner service is a separate tool called *Amber Collector* and will be available as open source under this name in the future. It is built on *Python* and provides (user) interfaces for performing and checking such scans of paper surveys. The whole scanning process, web surveys, etc. are part of the phase *data collection*. The results of this phase are the collected data as data flow.

Besides the printable, web, and smartphone surveys and the templates for the paper scanner, an automatically derived *verifier* is another important result of the questionnaire compiler services. The *verifier service* uses that verifier, which belongs to the phase of data preparation. It checks the collected data for malformations and unsoundnesses. This can happen for example if during the scanning process multiple answers for a single-choice variable were identified. The application of the verifier results in a verified data.

The verified data arrive at the *data preparation services*. At this time, the user has created a questionnaire model, has implemented it by the compiler as paper and web survey, and has verified the collected data. Now, the user wants to use the data for a report. Therefore, the necessary data has to be requested. Furthermore, the user wants to delete or modify open (text) answers, which may contain profanity or misspellings. The preparation services contain functionality to request, replace, and standardize data information. Naturally, it should not be able to modify the originally collected data. Therefore, each modification is done on a copy of the data information and is logged by the system. So it is possible that other researchers can reconstruct the modifications and are able to detect potential problems in other research. The same happens during the replacement of default, no choice, and other special values by user-defined values.

The replacements and computations of the data preparation services are done with *R*, a statistical programming language which is perfect for data handling. As shown in the figure, there is a bidirectional data flow from the data preparation services to the *OpenCPU R services*. The *R* services are like a pipe in which the data is transferred, processed, and the enriched data is sent back.

The *analysis and report services* receive the prepared data from the data preparation phase. However, those services have to know which analyses should be performed. Therefore, there is another service during the data preparation phase — the *report transformation service* — which translates a questionnaire model into a report model. To derive this information from the questionnaire model, the report transformation service maps the questionnaire to a new report, e. g., the items of the questionnaire to report parts and the representations of the items to computational representations. As a result of the service, there is a full specified standard report model for the entire questionnaire model. This is illustrated as the model flow to the *report model* in the figure.

Some users want to have more powerful and specialized computations in their reports, e. g., group comparisons, geographical maps, and regressions. With the *report designer services*, it is possible to modify the report model to one's need and remove items, change variables, add computations, add report items, add group comparisons, among other things.

The report model together with the collected and prepared data are combined in the *analysis and report services* for computations. As mentioned before, the report model contains the information about necessary calculations. The collected data are then used as input for the computations. By doing this, the report model is transferred into a linear topological order and the conditions on the edges are translated into set operations in order to filter the data. Since the services need a lot of statistical calculations, *R* is used again to perform the computations. All standard calculations are collected in a *R* package, but the user can freely add new functions to the analysis and report services.

The result of the analysis and report services are the computed values. Together with the report model, these values can be combined in a report. This is done by *report compiler services* similar to the questionnaire compiler services explained before. Instead of compiling the model to surveys, the report model is translated to reports for different output devices. In this case, a *device* means the medium for which the report is generated. For example, the compiler is able to create printable paper and online reports. For the former, the compiler services use a LaTeX service. They compile the entire report model and the values into a LaTeX document, which can then be compiled into a PDF or DVI document. This is where the report is finished.

C. Service-Orientation

The architecture uses a lot of concepts of MDE as mentioned before. Especially, the questionnaire and report compiler services of Figure 7 are strictly model-driven and do not allow modifications on the result without doing it on the model. Since the models are transferred within the architecture and a lot of statistical computations are done, the architecture contains possibly long-running and computationally expensive tasks. Such tasks should, however, not affect other tasks of the architecture, for example, the data collection of a survey. In other words, the tasks should *scale* and should be physically *separable*. With *Service-oriented architectures* (SOA) [11] such a separation and scalability is easier and, therefore, a good choice.

Another reason for using SOA is that important parts of our architecture follow ongoing research in the context of psychoinformatics and compiler construction. That means that some of the tasks have to be flexible, evolutionarily growable, and replaceable. By encapsulating the tasks as services, the tasks should become loose coupled and should have high cohesion [32] — the services are almost independent of other services. Changing the services but retaining their interfaces makes them replaceable. For these reasons, each task (module) of the architecture is defined as a service with its own interface. If the functionality of the service is shared with other systems or users, it will be provided as a *RESTful* web interface. Otherwise, it is a simple system interface for reasons of performance. The flows in Figure 7 indicate how the modules interact with each other. For

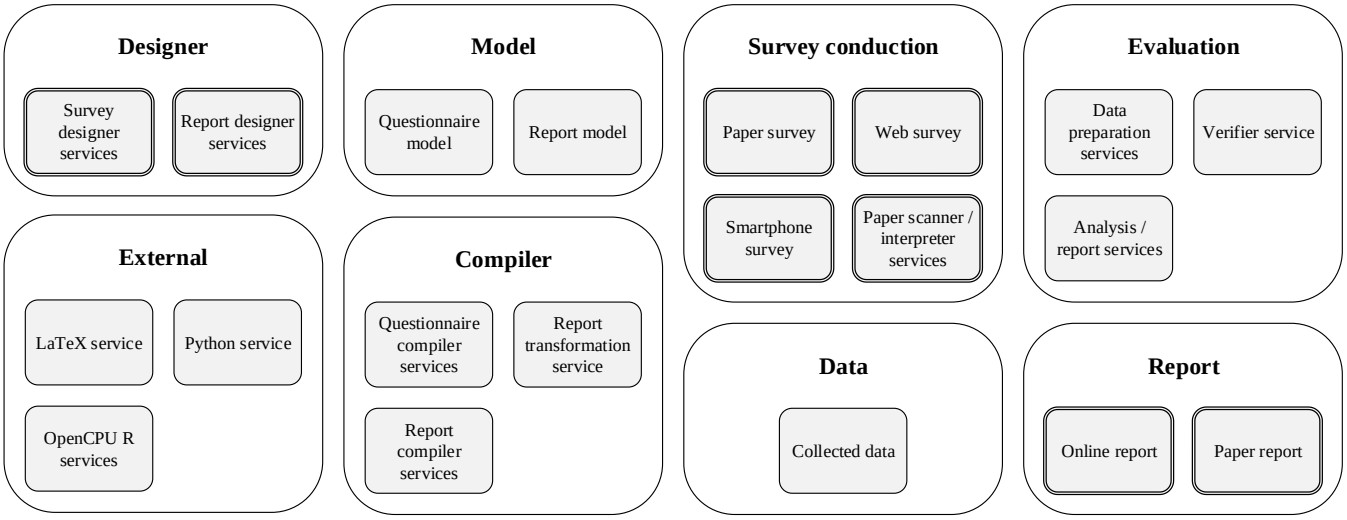

Figure 8. The integration layer of the *Coast* system.

example, to use *R* as a service, the *openCPU* [33] engine is used. OpenCPU makes it possible to use *R* as a RESTful web service.

Although the architecture is service-oriented, it is not a classic SOA. Some concepts have not been implemented in the architecture (yet). One missing component is a classic service repository. Right now, there is a repository for the different hosts where the web surveys are running. But all necessary system services are known in the architecture and do not have to be linked dynamically. Other services do not work via a network because of security and performance related issues.

Some of the services interact with the user. In order to achieve such communication, the architecture decomposes into server and client applications following a typical enterprise architecture. On the server, all the computations are done, the models are stored, the compilations are performed, and the collected data is handled. The client applications are mainly functional user interfaces allowing to talk with the server applications and to hide architectural and modelling details. Since the communication of the client to the server is done via HTTP, the whole architecture also describes a web application based on different modules.

It has become best practice during the development of the *Coast* architecture to maintain each module in separate projects and to build up the system based on them. This approach helped us to generalize modules, to define proper interfaces, to get loosely coupled functionality, and to reuse code whenever possible.

D. Integration Layer

Using services requires the *componentization* of the application into subsystems and modules (services) as described in Figure 7. In general, the architecture of *Coast* disintegrates into eight abstract layers: *designer*, *model*, *compiler*, *survey conduction*, *data*, *evaluation*, and *report* (illustrated in Figure 8). The collection of the different layers is called *integration layer* in SOA.

The *designer* contains the services belonging to the interaction with the user. Most of the business logic is configured with the designer. Therefore, it is the main application, which constructs the questionnaires and reports, starts the compilation of them, and initialize the analyses of the data. The survey and report designer services are part of the layer.

The *model* layer contains the structure of the questionnaire and report models and arranges their storing, for example, in a database.

The *compiler* layer with the compiler and report compiler/transformation services uses the model and configurations (done in the designer layer) to transform them either in other models or in documents, programs, or intermediate representations (IR). For example, the compiler produces the IR *liQuid* [34], which contains all necessary information about running a web survey. Furthermore, *liQuid* contains pre-compiled representations of each page, computed remaining pages, etc.

The *survey conduction* layer handles the data collection with the different kinds of surveys. This contains the paper, web, and smartphone surveys as well as the paper scanner and interpreter services. As part of the web surveys, there is a virtual machine (a survey engine). *liQuid* can be transferred to the survey engine, which executes it. Together, the compiler and survey conduction layers follow an old computer science principle: Compilation and execution. Since the engine can be separated from the rest of the architecture, it is possible to have multiple survey engine instances on different physical systems. As a result, the survey conduction becomes scalable.

The survey conduction produces data. The *data* layer handles the storing and processing of that data information. Furthermore, it allows the request and combination of different data sources.

The *evaluation* layer uses the report model, which offers computations, variables, and structures needed during evalu-

ation. The evaluation accesses the data layer to get the right data information. Furthermore, it stores its results back in the data layer.

There is an *external* layer in Figure 8, which contains the LaTeX, Python, and OpenCPU R services. Those services build wrappers around existing tools to integrate them into the architecture. They have exclusively functional interactions with the other layers. The services/modules with double lines in Figure 8 instead have also interactions with the user. They need user interfaces. For the paper surveys and reports the term "user interface" has a symbolic meaning. However, the web survey, online report, designers, and scanner services have high user interaction and, therefore, real user interfaces. The other services in the figure do not interact with the user. Their interfaces are functional.

V. Benefits and Problems of the Approach

The explained architecture used in our tool *Coast* is one possibility and implementation of a software architecture for surveys. In the previous sections, it was explained, *how* the data model and architecture look like and *why* both were constructed in this way. This section further describes benefits and problems emerged during implementation time, usage, and ongoing development.

The main benefit of the introduced architecture is the coverage of our requirements specified in Section II. The architecture allows the compilation of the questionnaires and reports to different devices, e. g., paper, web, and smartphone, using mostly the same infrastructure of algorithms (requirement DEV). It enables the addition of new kinds of question visualizations (time tables, sliders, etc.) since each item has an assigned representation in the data model (requirement QAD, also cf. Section III-B). If there is research about new visualizations, it can be easily adapted in the architecture. Although the reporting was not introduced in detail, the report model uses representations and abstract calculations to extend the analyses with new functionality for more complex statistical analyses (requirement AAD). Different surveys and data sources can be merged into a single report and new content can be added (requirement MUL). For questionnaires and reports, the mathematical and data models naturally define adaptivity as a graph (requirement ADA). Altogether, both models are highly flexible and cover many types of surveys and reports (requirement FLE).

As mentioned at the end of the last section, the architecture uses concepts of SOA. Therefore, almost all parts of the architecture are services. One main service is the survey engine executing compiled questionnaire models (*LiQuid* [34]). This engine is installable and executable also in private networks (requirement PRI). It can be hosted on different servers making it scalable. Since the engine can run on different servers from the other tasks, the processes do not interact and, therefore, do not reduce the performance (requirements DIS and SCA).

The consequent usage and the level of detail of the models during all phases allow to use these detailed information in each phase of a survey. For example, the report model results from a transformation of the questionnaire model. By doing this transformation, the report model gets the same variables, conditions, and codomains like the questionnaire. There is no missing information for the report. Since the questionnaire and report models are generalized and do not only fit to *Coast*, they could be used in other survey tools, too. These models can be stored for their documentation and reuse in future as well.

Other benefits regarding the introduced model of questionnaires (and reports) were shown in previous work [34] [35]. The former considers the erroneous multiple asking of the same variable in a questionnaire. It explains an algorithm based on static and dynamic analyses, which finds multiple occurrences of and assignments to the same variable on the same path. This could lead to the loss of information if a variable was wrongfully assigned to an item.

The latter work [35] reconsiders progress indicators in web surveys, whose calculation is based on an abstract version of the meta-model. This made it possible to define a general algorithm to calculate the progress in complex surveys with branches in paths.

An improvement during questionnaire creation is the visualization of the questionnaire model as a graph. It helps to keep the overview of all paths and variables as well as to maintain the central theme. The control-flow-graph-like structure shows the adaptivity and individual paths for groups of participants can be checked (by pre-assigning variables at construction). For example in a student survey, if a set of items should only be visible to Master students at different points of the questionnaire, the variable representing the degree can be set to "Master" to visualize the remaining paths.

A benefit of the graph-based model of reports is its advanced usage during the data analysis and reporting phases. The graph can be used to define wide varieties of item orders. A topological sorting helps to linearise these orders in a paper-familiar way. Furthermore, the graph can be used to produce code to only select data of the currently considered items. It reduces unnecessary data information for analyses.

The separation of the architecture in components and services is a great benefit during implementation. The previous monolithic architecture of the first version of our tool *Coast* decomposed into logically separable projects. These projects are more reusable in other contexts and are easier to maintain from our own experiences. One mentionable example is the reimplementation of the database done in the architecture without high reimplementation costs in other projects.

The separation into different projects makes it also easier to talk about the architecture in the team since each project has a clearly defined purpose and functionality, and a unique name to refer to it. A great advantage is the focus on the questionnaire and reports instead of their implementation. It matters *what* is done, not *how*.

The new architecture has issues too, naturally. Sometimes, the performance of the system is slower than in our first monolithic application. This comes from the increased overhead by using SOA. Furthermore, there are sometimes a lot of messages being transferred between the different services. Especially, the communication between the client and the server during the design of questionnaires and

reports is verbose.

The fine-grained models for the definition of questionnaires and reports result in large models of sometimes thousand to ten thousand objects. If a user wants to modify such a model within the web application, all those objects have to be taken from the database and have to be sent to the client. Sometimes, this took more than 30 seconds — which is unacceptable. Therefore, the pattern of *lazy loading* [36] objects had to be introduced in the architecture. When the client application tries to access an associated object, the system loads the required object from the server automatically if it was not already loaded before. Nevertheless, to allow a straight forward modelling and programming, the concept of promises [37] was extended to an ordinary if-then-construct making the programming of asynchronous services more intuitive.

Another disadvantage is the restriction of handling special cases within questionnaires and reports only via the data model although it appears sometimes faster and easier to implement them directly. Such special cases need the extension of the data model and, therefore, mostly more time. Examples are the representations of items in HTML. Sometimes these representations need only minor modifications to fit the desired visualization. However, since the modification of the representation may result in undesired side effects in other already implemented questionnaires, a new representation has to be introduced or the representation has to be extended by additional parameters.

There are other kinds of surveys which cannot be handled intuitively with our architecture, yet. For example, in computerized adaptive testing (CAT), the items of the test (survey) are not shown in sequence [38]. Their visibility depends on the answers of the previous items and they are randomly chosen based on a user-specific score. Most parts of CAT can be achieved with our architecture proposal. However, the randomly chosen selection of items and the highly connected structure of the items are not implemented yet and needs extensions of the current data models.

Although there are weaknesses and open issues on the current design, the benefits at practice prevail the issues. Since its introduction in our department, the software completely supports all phases of our surveys with high flexibility.

VI. Conclusion and Outlook

This paper described the realization of a survey system called *Coast*. The survey system keeps the interdependencies between the survey phases intact. Since the analysis and reporting phases of surveys are highly individual tasks and need a lot of information of the preceding phases, it was necessary to focus on the links between the phases.

The focus on the phases manifested in the paper as a detailed consideration of how questionnaires and reports are structured. This consideration took place both as a mathematical and a data model. The latter forms a DSL, which can be used to construct arbitrary questionnaires, reports, and other survey-related parts. Furthermore, the data model is the centre of the architecture of *Coast*. The architecture is, therefore, build up around the data model and describes how the models are transformed within the application. It also describes the necessary services, flows, and components.

In the end, the paper discussed the benefits and disadvantages of the proposed architecture as lessons learned. This was done especially regarding our advanced requirements on a survey tool, which were introduced too. There is no other survey tool, which fulfils all those requirements. This made this work necessary.

Since the *Coast* system is currently in an alpha version, the system is unpublished up to now. The future versions should be available to everyone via the web. For this purpose, the application has to reach a stable stage, and some of the concepts have to be extended. For example, the services and models used in the analysis and reporting phases can be used also for surveys conducted outside the *Coast* environment. This is possible by building a report model up from scratch without having a preceding questionnaire model. This report would receive the externally collected data as input. These data have to be described with the data model introduced in this paper. The concept of the user interface has to be extended to allow the description of variables and codomains subsequent to the data collection phase.

References

[1] T. M. Prinz, R. Bernhardt, L. Gräfe, J. Plötner, and A. Vetterlein, "Using Service-oriented Architectures for Online Surveys in Coast," in Service Computation 2018: The Tenth International Conferences on Advanced Service Computing, Barcelona, Spain, February 18–22, 2018. Proceedings, pp. 1–4.

[2] V. M. Sue and L. A. Ritter, Conducting Online Surveys, 2nd ed. Los Angeles, USA: SAGE Publications, 2012.

[3] J. Reinecke, Handbuch Methoden der empirischen Sozialforschung (Handbook Methods of Empirical Social Research). Wiesbaden, Germany: Springer, 2014, vol. 1, ch. Grundlagen der standardisierten Befragung (Basics of standardized Surveys), pp. 601–617.

[4] Electric Paper Ltd., "Survey Automation Software - EvaSys and EvaExam," Website, available: http://en.evasys.de/main/home.html, retrieved: February, 2019.

[5] Problem Free Ltd., "KwikSurveys: Make online surveys, quizzes and forms," Website, available: https://kwiksurveys.com/, retrieved: February, 2019.

[6] LimeSurvey GmbH, "LimeSurvey: the online survey tool - open source surveys," Website, available: https://www.limesurvey.org/, retrieved: February, 2019.

[7] SurveyMonkey, "SurveyMonkey: The Worldś Most Popular Free Online Survey Tool," Website, available: https://www.surveymonkey.com/, retrieved: February, 2019.

[8] QuestBack GmbH, "Startseite — Unipark," Website, available: https://www.unipark.com/en/, retrieved: February, 2019.

[9] Butter CMS, "Microservices for Startups," Open Access, https://s3-us-west-2.amazonaws.com/buttercms/ButterCMS+presents+MicroservicesForStartups.pdf, retrieved: February, 2019.

[10] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From Monolith to Microservices: A Classification of Refactoring Approaches," CoRR, vol. abs/1807.10059, 2018.

[11] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, and U. Zdun, Software-Architektur: Grundlagen - Konzepte - Praxis (Software Architecture: Basics - Concepts - Practice), 2nd ed. Heidelberg, Germany: Springer, 2009.

[12] T. Mika and M. Stegmann, Handbuch Methoden der empirischen Sozialforschung (Handbook Methods of Empirical Social Research).

Wiesbaden, Germany: Springer, 2014, vol. 1, ch. Längsschnittanalyse (Longitudinal Analysis), pp. 1077–1087.

[13] IBM United Kingdom Limited, "IBM SPSS Statistics - Overview - United Kingdom," Website, available: https://www.ibm.com/uk-en/marketplace/spss-statistics, retrieved: February, 2019.

[14] Microsoft, "Microsoft Excel 2016, Download Spreadsheet software — XLS XLSX," Website, available: https://products.office.com/en-gb/excel, retrieved: February, 2019.

[15] The R Foundation, "R: The R Project for Statistical Computing," Website, available: https://www.r-project.org/, retrieved: February, 2019.

[16] J. Bethlehem, "The TADEQ Project: Documentation of Electronic Questionnaires," in Survey Automation: Report and Workshop Proceedings, pp. 97–116, 2003.

[17] G. Chartrand and P. Zhang, Discrete Mathematics, ser. 1 edn. Long Grove, Illinois, USA: Waveland Press, Inc., 2011.

[18] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, ser. 3. Cambridge, UK: PHI Learning, 2010.

[19] M. Bühner and M. Ziegler, Statistik für Psychologen und Sozialwissenschaftler (Statistics for psychologists and social scientists), ser. 1. Auflage. Munich, Germany: Pearson, 2009.

[20] G. Brancato, S. Macchia, M. Murgia, M. Signore, G. Simeoni, K. Blanke, T. Körner, A. Nimmergut, P. Lima, R. Paulino, and J. Hoffmeyer-Zlotnik, Handbook of Recommended Practices for Questionnaire Development and Testing in the European Statistical System, 1st ed. European Statistical System (ESS), 2006, no. European Commission Grant Agreement 200410300002.

[21] S. S. Stevens, "On the Theory of Scales of Measurement," Science, vol. 103, no. 2684, pp. 677–680, 1946.

[22] J. Rost, Lehrbuch Testtheorie – Testkonstruktion (Textbook Test Theory – Test Construction), ser. Zweite, vollständig überarbeitete und erweiterte Auflage. Bern, Switzerland: Huber, 2004.

[23] M. A. Robinson, "Using multi-item psychometric scales for research and practice in human resource management," Human Resource Management, vol. 57, no. 3, pp. 739–750, 2018.

[24] H. Moosbrugger and A. Kelava, Eds., Testtheorie und Fragebogenkonstruktion (Test Theory and Survey Construction), ser. 2., aktualisierte und überarbeitete Auflage. Berlin, Germany: Springer, 2011.

[25] E. Martin, Encyclopedia of Social Measurement. Elsevier, ch. Survey Questionnaire Construction, pp. 723–732, 2005.

[26] J. B. Gregg, "Questionnaire Construction," Hospitality Review, vol. 7, no. 2, pp. 45–56, 1989.

[27] P. J. Pahl and R. Damrath, Mathematical Foundations of Computational Engineering: A Handbook, ser. 1. Auflage, F. Pahl, Ed. Berlin, Germany: Springer, 2001.

[28] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Pearson New International Edition: Principles, Techniques, and Tools, 2nd ed. Essex, UK: Pearson Education, 2013.

[29] E. J. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley, 2003.

[30] R. Soley and OMG Staff Strategy Group, "Model Driven Architecture," Object Management Group, Needham, USA, White Paper Draft 3.2, 2000.

[31] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse Modeling Framework, 2nd ed., ser. The Eclipse Series, E. Gamma, L. Nackman, and J. Wiegand, Eds. Boston, USA: Addison Wesley, 2008.

[32] E. Yourdon and L. L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 1st ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 1979.

[33] J. Ooms, "The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns," Computing Research Repository (CoRR), vol. abs/1406.4806, pp. 1–23, 2014.

[34] T. M. Prinz, L. Gräfe, J. Plötner, and A. Vetterlein, "Statische Aanalysen von Online-Befragungen mit der Programmiersprache *liQuid* (Static Aanalysis of Online Surveys with the Help of the Programming Language *liQuid*)," in Proceedings 19. Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2017, Weimar, Germany, pp. 59–70, September 25–27, 2017.

[35] T. M. Prinz, R. Bernhardt, J. Plötner, and A. Vetterlein, "Progress Indicators in Web Surveys Reconsidered — A General Progress Algorithm," in ACHI 2019: The Twelfth International Conference on Advances in Computer-Human Interactions, Athens, Greece, February 24–28, 2019. Proceedings, pp. 101–107.

[36] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, Patterns of Enterprise Application Architecture, 1st ed., M. Fowler, Ed. Boston, USA: Addison Wesley, 2003.

[37] B. Cavalier and D. Denicola, Promises/A+ Promise Specification, Open Access, Promises/A+ organization Std. 1.1.1, 2014, available: https://promisesaplus.com/, retrieved: January, 2018.

[38] W. J. van der Linden and C. A. W. Glas, Eds., Computerized Adaptive Testing: Theory and Practice. Netherlands: Kluwer Academic Publishers, 2000.