Modeling a Tool for Conducting Systematic Reviews Iteratively

Brice Bigendako and Eugene Syriani
DIRO, Université de Montréal, Montréal, QC, Canada

Keywords: Systematic Literature Review, Model-driven Engineering, Automatic Installation.

Abstract: Conducting systematic reviews (SR) is a time-consuming endeavor that requires several iterations to setup right. We present ReLiS, a tool to automatically install and configure SR projects to conduct them collaboratively and iteratively on the cloud. ReLiS is engineered following a model-driven development approach. It features a domain-specific modeling editor tailored for researchers who perform SR and an architecture that enables on-the-fly installation and (re)configuration of multiple concurrently running SR projects.

1 INTRODUCTION

Publishing systematic reviews (SR) in areas of science and engineering (e.g., software engineering, medecine, social sciences) is considered essential in research communities (Thomas et al., 2010). Such secondary studies must be performed in a systematic and repeatable way, as proposed by guidelines (Kitchenham and Charters, 2007). SR are difficult and time consuming to conduct. In most cases, researchers perform the process manually, with no tool support (Imtiaz et al., 2013). A few SR tools exist today (e.g., StArt, Parsifal, and EPPI-Reviewer) to help manage references, perform screening, collect metadata, and report results of the study.

A recent community study identified a pressing need for better tool support (Hassler et al., 2016). One key feature of SR tools is the ability to cope with the collaborative and iterative process of SR. Hence, it is crucial that an SR tool can modify the SR protocol on-the-fly and reconfigure itself. Web applications offer a convenient technological foundation for such flexibility. However, they mainly lack flexibility on several ends: no iterative process is supported by piloting SR and they have a rigid configuration of projects that are hard to change (adding validation phases, advanced data extraction forms). This is where model-driven development (MDD) can help by automatically synthesizing applications from a domain-specific model (Kelly and Tolvanen, 2008).

In this paper, we follow an MDD approach to automatically configure and reconfigure an SR tool in order to reach the desired protocol and adapt to the needs of the secondary study. We have developed the framework ReLiS[1], which offers an integrated environment based on a domain-specific language (DSL) for installing and configuring SR projects so that a group of researchers can conduct a SR collaboratively using an application customized to their desired research topic. The contribution of ReLiS is the holistic solution to not only manage the SR process by tracking progress and storing related data, but also in its flexibility to configure SR projects.

In Section 2, we briefly outline how SR are conducted and how this is performed in ReLiS. In Section 3, we explain how we use domain-specific modeling and MDD to define configurations for SR projects. Then, in Section 4, we describe the architecture that enables the deployment of online SR projects in ReLiS. In Section 5, we present a preliminary evaluation of ReLiS. We discuss related work in Section 6 and conclude in Section 7.

2 CONDUCTING SYSTEMATIC REVIEWS

SR is a research strategy used to extract new knowledge from existing studies. It follows a well-defined process with a rigorous planning and methodological execution to ensure unbiased results with a repeatable and auditable study. An SR can take the form of a systematic literature review (SLR) to evaluate and interpret all available studies relevant to a particular research question, topic area, or phenomenon of interest (Kitchenham and Charters, 2007). It can

[1] http://relis.iro.umontreal.ca/.

Bigendako, B. and Syriani, E.
Modeling a Tool for Conducting Systematic Reviews Iteratively.
DOI: 10.5220/0006664405520559
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 552-559
ISBN: 978-989-758-283-7

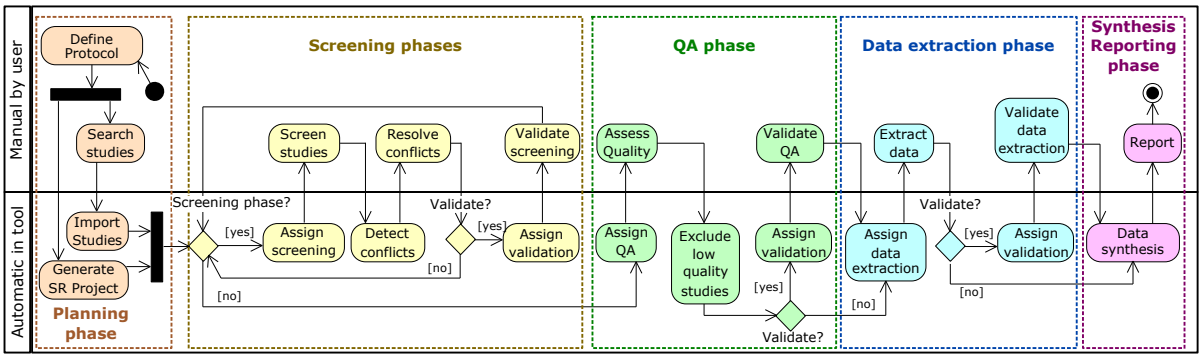

Figure 1: SR process in ReLiS.

also take the form of a systematic mapping studies (SMS) to provide a wide overview of a research area by classifying and performing a thematic analysis on the topic (Petersen et al., 2008). Although these two types of secondary studies have different intentions, they share many commonalities in the way they are conducted. The process of conducting SR is separated into three phases consisting of actions to be performed iteratively. The three phases are *planning* the study by developping the protocol to follow, *conducting* the review by screening primary studies and extracting the relevant data, and *reporting* the results of the analysis of the data.

2.1 SR Process in ReLiS

Figure 1 illustrates a typical workflow to use ReLiS following proposed guidelines. The user starts by planning the SR and sets up a project in ReLiS. He creates a SR project by defining a configuration model (detailed in Section 3) that automatically installs the project on the cloud: a web application on the web server and a dedicated database. The model plays the role of the protocol of the SR and customizes the steps of how the review will be conducted. As SR are collaborative endeavors, the user can add participants who will be conducting the review. Different roles can be assigned, such as reviewers, validators, or project managers, giving them increasing access rights.

Researchers search for primary studies outside ReLiS, by querying digital libraries directly. After filtering them, they can import them in their BibTeX, EndNote or CSV formats. ReLiS stores the meta-information of each paper (e.g., title, abstract, author, venue, year) and a link to the full text. Each imported study is tagged with its source and search strategy.

As soon as papers are in ReLiS, the participants can start screening the corpus and decide which paper to include or exclude, as depicted in Figure 2. Each paper can be assigned automatically and randomly to a fixed number of reviewers. ReLiS automatically detects conflicting decisions between reviewers of the same paper, so that they can resolve them according to the strategy defined in the protocol. ReLiS supports multiple screening phases that each show specific meta-information.

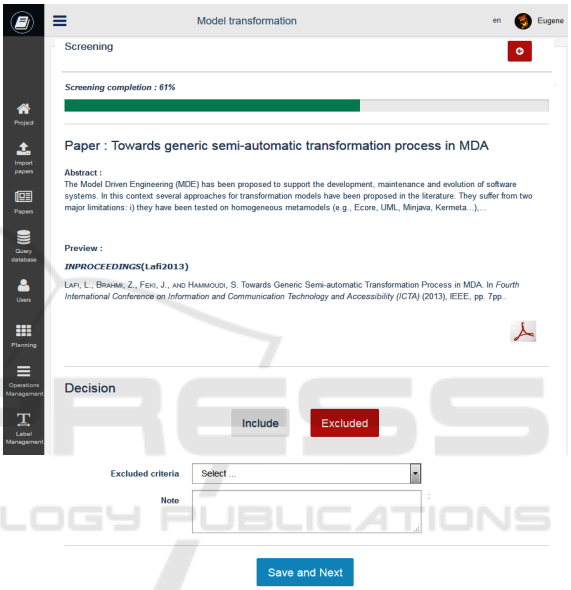

Figure 2: Screening form.

Following the questions and answers defined in the configuration model for quality assessment (QA), ReLiS generates forms to assess the quality of included studies. Studies are assigned to participants with the validator role for assessment. ReLiS automatically calculates the score of each study and flags low quality ones for exclusion.

ReLiS automatically generates the data extraction forms from the configuration model. Reviewers read through each paper and fill the online form to classify or highlight relevant information of each retained paper, as depicted in Figure 3. Like for screening, papers can be automatically assigned to reviewers for data extraction.

From this information, reviewers can explore the

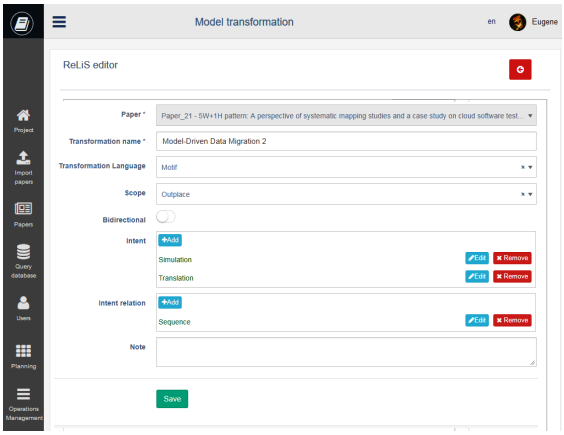

Figure 3: Generated data extraction form from Figure 5.

quantitative results in the form of graphs, charts and tables in ReLiS. All the extracted data can be exported to a CSV file to be used in more advanced statistical tools (e.g., SPSS, Mplus, R, Excel). ReLiS can also generate a partial report of the protocol: all the steps effectively conducted during the SR.

If needed, ReLiS allows validators to validate results of the screening, QA or data extraction. It selects a percentage of the processed papers at the corresponding phase and randomly assigns them to another reviewer for validation. The tool tracks the progress and reports basic statistics for each phase, rendered as tables and plots that can be exported for further analysis and used to produce the report of the review. Furthermore, ReLiS supports an iterative process: new papers can be imported at any time, project managers can disable or re-enable a completed phase, and they can modify the configuration of the project at any time (e.g., modifying exclusion criteria or the classification scheme). ReLiS supports the conduction of SLR or SMS. Screening and QA are optional, for example if the researcher only wants to collect data from a known set of studies.

3 FLEXIBLE CONFIGURATION OF SR PROJECTS

We motivate the use of MDD to configure SR projects and present how we applied it in ReLiS.

3.1 MDD for SR Project Configuration

Our goal is to give the flexibility to users to configure and install projects by themselves without prior system administration skills. This is particularly useful for SR tools when the SR process is iterative, because it is often impossible to plan the right protocol correctly from the beginning. The protocol needs to be adapted as special cases are encountered during screening and data extraction. First conducting a *pilot study* is usually recommended to make sure all participants have the same understanding of the protocol (Staples and Niazi, 2007).

Existing tools allow configuration via wizards which is good for parameterizing but does not offer flexibility to change the semantic of the project. By using MDD, the process focuses on what varies in the general guideline to a particular SR protocol. With MDD, we can simplify the configuration process of an SR project, by letting users specify a configuration model that abstracts the specifications common to all SR projects as well as the details related to underlying implementation and infrastructure. Consequently, users can perform minimal changes to the model to adapt the configuration, but this may have a significant impact on implementation (data base, code and available functionalities), which remains seemless to the user thanks to automatic code generation.

For these reasons, we opted for a MDD approach by defining a DSL, which allows the user to edit how the components specific to SR projects shall be configured. To support this—possibly iterative—process, users can customize the SR procedure in a controlled way. The steps common to any SR procedure are built-in ReLiS, with some level of customization, such as the way conflicts are detected and resolved during the screening phase. Furthermore, some steps are specific to the particular SR project and to the research question or area, e.g., the classification scheme and data extraction form.

3.2 Configuration DSL

In the planning phase, the SR project is configured in ReLiS through a DSL. We define the metamodel of the DSL presented in the class diagram in Figure 4. It offers customizations points to the process outlined in Section 2.1. The central entity is the SR `Project` identified by a unique `name`. It is composed of four main elements.

When a `Screening` step is desired, the user can configure how papers are assigned to reviewers, inclusion and exclusion criteria. He can also define what happens in case of conflict decisions among reviewers of the same paper. Two types of conflicts are supported: either if one reviewer excludes a paper that another has included, or if the exclusion criteria do not match. A screening conflict of a paper is resolved either by having a unanimous decision among reviewers or if the majority agree. If the validation

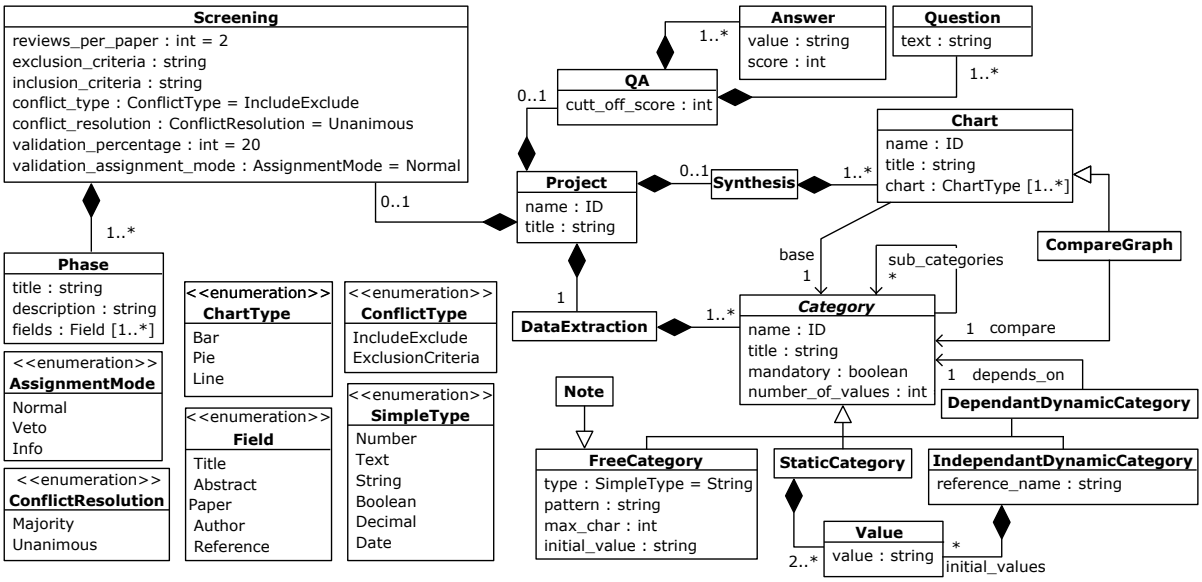

Figure 4: Project configuration metamodel.

of a screening phase is enabled, the user can define what proportion of the excluded papers will be validated randomly. He can also state how the decision of the validator is to be treated: as another voting reviewer, whether his decision overrides all others, or it is simply used to notify the other reviewers. It is possible to configure the screening iteratively into phases where specific (meta-)information of each paper will be available to the reviewer. Screening is optional in ReLiS to support the use case when the user has already selected his papers and is only interested in extracting data from them.

When a `QA` step is desired, the user has to define the `questions` and the possible `answers` (associated with a `score`) to build the QA check-list for each included paper. A `cutt_off_score` identifies the minimum score for which papers are considered of acceptable quality.

The `DataExtraction` defines what categories to collect in the form of each paper. For instance, this corresponds to the classification scheme of a SMS. The metamodel supports four type of categories. A `FreeCategory` allow users to freely enter a value, subject to a specific type (text, boolean choice, numerals, etc.) or to some additional constraints (length, regular expression, etc.). A `StaticCategory` offers to choose a value from a predefined list of values, the list cannot be updated while conducting the review. An `IndependantDynamicCategory` is like a `StaticCategory`, but the list can be updated during the SR process. A `DependantDynamicCategory` is similar to the former, but the allowed values must come from values entered for another category. Each category is identified by it `name`. It can have a single or multiple values, and can also be mandatory to fill in the form. A category be more complex and contain subcategories, in which case a distinct form will be generated for it.

Finally, the project may specify a data `Synthesis` to define how the extracted data will be synthesized into charts. Various chart types are supported. A chart may render data from a single category or plot the relation between data from different categories.

3.3 SR Project in ReLiS

Figure 5 shows the online editor in ReLiS of the configuration model that was used to generate the project for a SMS on model transformation. That model is an instance of the metamodel presented on Figure 4. We chose to use a textual concrete syntax because the configuration follows a linear layout (following each phase) and to improve editing of the model (e.g., copy, paste).

In this SMS, the study selection shall be conducted in two screening phases (lines 8–10). A first phase screens papers according to their title only, followed by a second phase where the title, abstract and a link to the full content of the paper is available. Figure 2 shows the screening form generated for the second phase.

There is also a QA phase (lines 12–15) with two question and three possible answers per question for each paper. If the sum of the answer values is below 5 for a paper, then it will be considered low quality.

The following section configures the data extraction form generated in Figure 3. Line 18 generates a text field to collect the name of a model transfor-

Figure 5: Project configuration model in ReLiS in its textual concrete syntax.

mation with at most 100 characters. It is a mandatory field (with the '*') and can only have a single value ('[1]'). The transformation language field on line 19 is an `IndependantDynamicCategory` prepopulated with a list of four language names. Line 20 generates a drop down list (`StaticCategory`) to collect the scope of the transformation which must be one of three choices. The intent relation field has subcategories (between '{...}') (line 25). It is allows for an arbitrary number of values ('[0]'). On line 26, it has a subcategory *Intent 1* that is a `DependantDynamicCategory` for which the value must be one of those entered for the *Intent* field on line 22. ReLiS comes with predefined categories that can be reused, such as `note` (line 30).

The data synthesis is specified in lines 34–36. The first statement will synthesize data extracted for the scope into a bar chart and a pie chart. The second one will synthesize data for the scope on the Y-axis per year on the X-axis.

3.4 Generation of a Configuration

We implemented the DSL with a textual concrete syntax using Xtext (Bettini, 2013). The model editor is completely integrated inside ReLiS thanks to DSLFORGE (Lajmi et al., 2014) which generates online lightweight text editors from an Ecore metamodel and an Xtext grammar.

To install projects in ReLiS, we developed a template-based code generator in Xtend (Bettini, 2013) to produce the installation file from the model, as illustrated in the bottom part of Figure 6. This process initiates the installation of the project in ReLiS. The presented approach ensures a natural continuum to the user who can operate the tool for its intended purpose (i.e., conducting SR), as well as instantly install new projects or reconfigure existing ones.

4 ARCHITECTURE TO RECONFIGURE PROJECTS

Apart from supporting SR activities, a tool for SR has to allow collaboration among researchers and manage tenancy of multiple SR projects.

Common web content management systems (CMS), e.g., Joomla, follow a Model-View-Controller (MVC) architecture. However, the traditional MVC implemented in CMS has two issues. First, as pointed in (Priefer et al., 2016), this architecture brings a certain overhead and schematically redundant code

which requires more effort in manually developing these extensions. Second, it requires to manually install the extensions and link them to the CMS. This means that every time one needs to change a functionality that is beyond simple user interface customizations, the user needs to re-install the extension. To solve these issues, we implemented a dynamic MVC architecture, illustrated in Figure 6.

4.1 Dynamic MVC Architecture

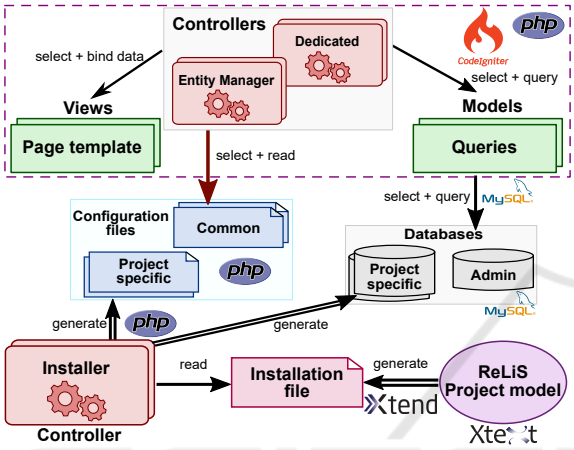

Figure 6: Architecture for dynamic MVC.

On top of models, views and controllers, we added the concept of entity in ReLiS, which represents the instances managed by the application (papers, users, authors, etc.). To ensure reusability and maintainability, a particular type of controller is dedicated to managing entities: the entity manager. There is one entity manager controller for each operation, such as adding, modifying, removing, listing, or viewing the details of an entity, parameterized by the entity type. For example, a controller is responsible for viewing the details of the meta-data of a paper (Figure 2), another one for modifying the classification of a paper (Figure 3), and a last one for listing the results of the classification (e.g., charts). The specificities of entities are encoded in entity configuration files.

An *entity configuration* specifies the characteristics of an entity. It defines its attributes (e.g., name, title), the models and stored procedures to invoke when retrieving, adding or updating data, and the way data will be displayed in web pages with the navigation logic between entities, i.e., the views.

What makes this MVC architecture dynamic is that the semantics of controllers is defined in PHP configuration files with controllers implemented as generic PHP classes making them independent from the entities they operate on: the same controller class can list all papers and all authors. There are other controllers dedicated to specific operations, such as the installation of projects and user authentication.

The advantage of this architecture is that it allows to change the application behavior by just using configuration files. To maximize reuse and modularity, the architecture of ReLiS contains components common to all SR projects and components specific to individual ones. Common configurations, which are built-in ReLiS, include, for example, how papers and users are configured. Project-specific configurations include how the data extraction form is configured. Although this distinction is useful conceptually, the architecture does not distinguish them and treats them alike. This facilitates the integration of new configurations and databases without having to modify any of the existing code. Furthermore, the advantage of using an interpreted language like PHP prevents the need to re-compile every time a new component is added.

4.2 On-the-fly Installation of ReLiS Projects

ReLiS is able to host multiple SR projects operated concurrently. To add a new SR project in ReLiS, the user defines a project configuration model (c.f. Section 3) which is automatically generated as an installation file. The latter defines entity types and their attributes on which entity controllers operate. The installation file also describes what information will be rendered on the screen, constrains the domain of each attributes, and defines information relevant to the database that will store the data of that project. As depicted in Figure 6, the *installer* is a controller that generates the configuration file corresponding to the installation file it reads in. It also creates a new database for the project, with tables and columns corresponding to entities, and the necessary stored procedures.

ReLiS allows a user to make modifications to a project while in operation. The presented architecture makes it relatively easy to add new entities or attributes (e.g., adding a new category in the data extraction form). However, deletion may corrupt the data already present in the application. The policy we adopted for is to only delete elements from a project if they do not contain any data yet. Otherwise, the element is deactivated but the data it contains is still retained in the database. This also allows to revert certain actions in case of erroneous manipulation. Edition of elements, such as renaming, are considered as deletions and additions. Nevertheless, we could detect changes that do not corrupt the data, such as augmenting the length of a string in a simple cate-

gory. However, performing such detections strongly couples ReLiS to the underlying database used, which hampers its portability.

The tool logs all major operations performed by the user, so he can revert them, e.g., remove a specific set of wrongly imported papers or clear the results of a screening phase, especially when there are a lot of disagreements in the study selection among the reviewers.

5 EVALUATION OF RELIS

We have performed a preliminary informal qualitative evaluation of the correctness of the implementation and the usefulness of the architecture decisions in ReLiS. In the context of a graduate course on empirical methods in software engineering at our university, all students are required to conduct a SMS on different research topics as part of their project. In the Fall 2016 edition, there were in total 8 groups of 3 to 4 students with no previous experience in conducting SR. Students were asked to use ReLiS for their SMS project. This allowed us to have test subjects to verify that the tool was functionally correct. This also gave us the opportunity to assess whether the category types, to generate the data extraction forms, built-in ReLiS are sufficient to cover all eight SMS. All groups used every type of category, but none applied advanced constraints.

We were also interested in determining how on-the-fly installation and configuration of projects was used. For that, we extended ReLiS to log every installation of projects, while tracking the modifications performed in the project-specific model. We noted that all re-installations were aimed at improving the classification scheme, since the screening procedure should be fixed from the beginning and changing it during the process may bias the SR. Students found that automatically re-installing their project was very helpful. This was expected because, during the learning process, they required several iterations before arriving at the desired classification scheme. The most positive feedback was that they were able to re-install a modified project without loosing the information on papers they had already classified. However, we noted a larger amount of deactivated categories than anticipated. We discovered that it was mainly because of misunderstanding the syntax of the DSL.

6 RELATED WORK

There are several tools designed to support the conduction of SR that were surveyed in (Marshall and Brereton, 2013; Al-Zubidy et al., 2017). Some tools specialize only in some desirable SR features, such as reference management and text-mining techniques to help extract relevant data from papers (Felizardo et al., 2010). Other tools are more specific to how SR are conducted in software engineering and aim to support the whole SR process. However, they still have important features not properly supported. *Parsifal*[2] is the most similar tool to ReLiS. It is an online tool that allows researchers to conduct only SLR collaboratively. However, it only provides simple category types for the data extraction forms and does not support, for instance, dependent lists or subcategories. *SLRTOOL* (S. Barn et al., 2014) also supports the whole SR process, but does not allow for multiple screening phases and requires adding studies manually. The key difference of all these tools and ReLiS is the flexibility of modifying the configuration of the SR procedure on-the-fly, which makes it able to pilot reviews. It is the only tool that uses MDD to adapt the tool to a specific SR by customizing the whole SR process and to generate the most advanced data extraction forms thanks to that DSL.

(Barat et al., 2017) have used MDD to support SR. They designed a DSL to represent research literature, with a metamodel to describe the core concepts of the review and associated process. However, their DSL is only used to collect the data of the SR process, not to generate a tool like ReLiS.

Related to the architecture of our approach, CMS are very generic frameworks that heavily rely on extensions developed outside the tool environment, as discussed in Section 4. ReLiS project configuration models are designed within ReLiS which requires no client-side installation of any tool, as it runs completely in a web browser, and projects are installed automatically. Base platforms to build web applications (e.g., Joomla and WordPress) support easy installation of custom extensions. However, their development requires the user to install the appropriate tooling and programming environment. Also, they are very generic and provide too many options that are not used in specific application domains, such as SR. The closest work to ours in using MDD for updating exiting application with on-the-fly installations is JooMDD (Priefer et al., 2016). It relies on a DSL where models get automatically generated as Joomla extensions, in order to raise the level of abstraction for web extensions development.

[2]https://parsif.al/

7 CONCLUSION

In this paper, we present ReLiS, a framework to automatically install and conduct SR projects collaboratively on the cloud. Its generic and dynamic architecture allows users to install projects during the process of a SR without manual intervention. Using MDD, a web-based modeling editor specific to SR empowers users to directly configure and install their project by themselves. Compared to other existing tools, ReLiS covers the most features, but lacks integrated search of studies, automated text analysis, and keyword extraction. It is freely available to be used online.

Our goal is to make this SR tool accessible to non-computer science researchers. We therefore plan to provide a more intuitive syntax for the configuration DSL and improve the feature coverage of ReLiS.

REFERENCES

Al-Zubidy, A., Carver, J. C., Hale, D. P., and Hassler, E. E. (2017). Vision for SLR tooling infrastructure: Prioritizing value-added requirements. *Information and Software Technology*, 91:72–81.

Barat, S., Clark, T., Barn, B., and Kulkarni, V. (2017). A model-based approach to systematic review of research literature. In *Innovations in Software Engineering Conference*, pages 15–25.

Bettini, L. (2013). *Implementing Domain-Specific Languages with Xtext and Xtend*. Number 2. Packt Publishing.

Felizardo, K., Nakagawa, E., Feitosa, D., Minghim, R., and Maldonado, J. C. (2010). An Approach Based on Visual Text Mining to Support Categorization and Classification in the Systematic Mapping. In *EASE*, pages 34–43. British Computer Society.

Hassler, E., Carver, J. C., Hale, D., and Al-Zubidy, A. (2016). Identification of slr tool needs–results of a community workshop. *Information and Software Technology*, 70:122–129.

Imtiaz, S., Bano, M., Ikram, N., and Niazi, M. (2013). A tertiary study: Experiences of conducting systematic literature reviews in software engineering. In *EASE*, pages 177–182. ACM.

Kelly, S. and Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Lajmi, A., Martinez, J., and Ziadi, T. (2014). DSLFORGE: Textual Modeling on the Web. In *Demonstrations at MODELS*, volume 1255. CEUR-WS.org.

Marshall, C. and Brereton, P. (2013). Tools to Support Systematic Literature Reviews in Software Engineering: A Mapping Study. In *ESEM*, pages 296–299. IEEE.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. In *EASE*, pages 68–77. British Computer Society.

Priefer, D., Kneisel, P., and Taentzer, G. (2016). JooMDD: A Model-driven Development Environment for Web Content Management System Extensions. In *ICSE Companion*, pages 633–636. ACM.

S. Barn, B., Raimondi, F., Athappian, L., and Clark, T. (2014). Slrtool: A tool to support collaborative systematic literature reviews. In *ICEIS*, volume 2, pages 440–447. SCITEPRESS.

Staples, M. and Niazi, M. (2007). Experiences using systematic review guidelines. *Journal of Systems and Software*, 80(9):1425–1437.

Thomas, J., Brunton, J., and Graziosi, S. (2010). EPPI-Reviewer 4.0: software for research synthesis. Tech report, University of London, EPPI-Centre, Social Science Research Unit, Institute of Education.