Abstract

Web applications are widely used due to the facilities they offer such as social interactions, easy access to information or even as enterprise business point. Along with all these utilities, traditional desktop applications are being gradually converted into web applications as it also brings advantages such as: easier support and maintenance; compatibility with a greater number of devices such as smartphones, laptops, tablets; ease of updating content. However, there are adjoining difficulties in web development, namely: need to know several languages and frameworks, security mechanisms, databases structure and responsive design.

This dissertation aims to develop a "RSL2WebApp" tool in order to generate source code for web applications requiring only a System Requirement Specification (SRS) from the user. For SRS it is used RSL which is a strict, consistent, and easy-to-use language for SRS that aims to reduce inconsistencies and reduce the number of errors during system specification. In the chapter Evaluation, it can be observed that starting from a few lines of specification of a system in RSL, it is generated a whole web application ready to run, making the web development process much easier and faster. In this report will be introduced several topics related to the development of web applications and the automation of it. It also explains in detail the operation of RSL2WebApp and the way in which a requirements specification is made in RSL.

Keywords: Aplicações Web, RSL, Model Driven Engineering, Angular6, ASP.NET Core 2, Xtend, Xtext

1. Introduction

The use of web applications is present in everyday life, as it facilitates various tasks.

However, the development of this type of web application can be a very challenging and time-consuming process since it is necessary to gather knowledge from various technical areas.

Due to all these difficulties involved in the development of web applications, a tool that is capable of generating source code for this type of web applications, can be helpful reducing complexity of the development of web application and be time saving.

RSL2WebApp is a tool developed in this work which reduces the complexity of web application development as it generates the whole source code for the desired web application. RSL2WebApp relies only on a System Requirements Specification (SRS) defined by the user. SRS is the development of a document describing the requirements and characteristics of a system. It provides a shared view among all key stakeholders during the design and implementation of a system. However, in general, the most common and preferred way to represent system requirements is through the use of Natural Language (NL), which has introduced some problems (as explained later). One solution to mitigate these problems is through the use of the RSL [4] language for requirements specification, which also facilitates the automation of some tasks related to the analysis and validation of requirements in RSL.

2. Objective

The main goal of this project is to develop a tool, called "RSL2WebApp", which is capable of generating source code for web applications from SRS defined with RSL. The web application generated will consist of an interface where the user can interact with and a server that contains all the logic together with the entire database structure that will provide the data that the user may be interested in obtaining. The interface will be built using Angular6¹ and the server will be built with the framework ASP.NET Core ². Xtend³ is used for code generation and parsing the requirements specification.

RSL2WebApp processes the SRS defined with

¹https://angle.io/
²https://docs.microsoft.com/en-us/aspnet/core/
³http://www.eclipse.org/xtend/programming
RSL through Xtext and Xtend technologies, and generated the final code for both client-side and server-side, as specified by the user.

Taking into account what was mentioned above, the objectives of this dissertation are:

- Research and learning in general knowledge of support technologies for Web applications; Learning RSL; Learning Xtend and Xtext technologies
- Definition and implementation of models for code generation for Angular6 and ASP.NET Core technologies, using Xtext and Xtend technologies, based on the RSL system specifications
- Validation and evaluation of the results obtained, based on concrete scenarios.

3. Structure
The remainder of this document is organized as follows;

- The Chapter Context provide the scope of this dissertation where some concepts are adopted and introduced such web applications, Requirements Engineering, Domain Specific Languages, the RSL initiative and Model Driven Engineering.
- The Chapter Support Technologies, presents some of the most relevant technologies and tools within the context of this project. Here are some of the most popular web development tools, both client-side and server-side, along with their pros and cons. Likewise technologies for generating code will also be discussed in this chapter. Subsequently, the choices for the technologies used in this project are enumerated and explained, taking into account the main objective. In addition, it shows an overview of RSL2WebApp, with a detailed description of how all parts are linked, namely how to transform the RSL specification to source code.
- In Chapter RSL2WebApp Approach, the QuickApp template project is presented, which serves as the startup project for RSL2WebApp. It is also explained how RSL2WebApp reads and parses the SRS in RSL and how the source code generation is done. Finally, it is explained how to define a SRS using RSL.
- The Chapter Evaluation, explains how the solution will be tested and evaluated.
- Finally, the chapter Conclusion, presents the conclusion of this work, with the negative and positive points.

4. Context
In this section, relevant subjects are introduced regarding the context of this dissertation.

4.1. Web Applications
A web application allows for a higher level of interaction and is intended to provide a user experience similar to that of a desktop application, including dynamic content.

Web applications are easily accessible from any device (tablet, smartphone, laptop, etc.) with Internet access using a web browser. In contrast to traditional desktop applications, which need to be pre-installed before they can be used, web applications can be accessed from any device as long as they have Internet access and a web browser. Some of the advantages of web applications are: greater availability; accessibility by a variety of different devices (smartphones, laptops, tablets, etc); easier maintenance and support; credibility in the business branch with an official web application.

4.2. Single Page Web Applications (SPA)
A SPA is a web application based on the "single web page" model, with the main purpose of offering a user experience similar to desktop applications [2].

In contrast to a tradition multi page application, a SAP usually all required code (HTML, JavaScript, and CSS) either is fetched with a single page load, or is loaded dynamically and added as needed. This causes that, whenever it is necessary to change the content of the page, the browser itself will handle the new rendering with JavaScript.

4.3. Model-Driven Engineering (MDE)
MDE is a software development methodology that focuses on the creation of abstract models that represent a partial and simplified view of a system. Generally, it is necessary to create several models to represent and better understand the system being studied. The MDE considers domain models as first-class entities and its main objective is for the modules to guide the whole development process (system design, code generation), which brings advantages such as product quality improvements, increased productivity and communication between stakeholders.

Originally, numerous techniques and modeling languages were proposed essentially with the main objective of obtaining a common and coherent understanding and vision of the system under study and, consequently, facilitating the communication between the interested parties [5].

However, in recent years a new methodology has emerged, not only considering models as documentation artifacts, but as central artifacts in the
software engineering process [5].

This new trend of approaches has brought more benefits beyond those offered by methodologies previously proposed, as it also allows the creation of automatic execution of software systems based on models that use complex techniques such as meta-modeling, model transformation, code generation, or model interpretation [5]. These techniques are used in this project for the code generation based on models.

4.4. Requirement Engineering (RE)

Having a clear and concise idea of the problem domain is essential to succeed during the development of a project.

Before beginning to develop the solution, there must be adequate and detailed documentation, as the success and cost-effectiveness of a software system development strongly depends on this aspect [1]. This activity concerns the area of RE. The RE area consists of several tasks/activities to provide a shared view/idea of the system among project stakeholders [4]. Although all activities are important, this project gives special attention to SRS activity. SRS is a document that describes several technical concerns of a software system (such as system requirements and user) [4]. In this context, a requirement is a description of what the system is capable of doing or a feature of the system. This document is used throughout the development process to facilitate communication between all stakeholders as well as a shared vision of the system. A good SRS is essential in many respects such as;

- Estimate of budget and schedule
- Agreement between client and company
- Fewer defects in requirements and final product
- Assurance of system characteristics and consequently less work to correct the unnecessary/wrong characteristics
- Greater customer and team member satisfaction

4.5. Domain-Specific Languages (DSLs)

DSLs are programming languages or languages of specification, specialized for a specific domain problem. These languages can not be applied to all kinds of problems, unlike a Generic Programming Language (GPL), which are applicable in all domains and can solve any kind of problem. On the other hand, if the problem domain is covered by a specific DSL, the problem is solved more easily and effectively. In the context of this project, we can imagine that a NL, for example English, is a GPL that can be used for any problem, and a more controlled language being a DSL that solves a specific problem, in this case the production of SRS.

4.6. RSLingo Approach

ITLingo is a long-term initiative to research, develop and apply rigorous specification languages [6] in the context of IT. For this, ITLingo mainly considers the following areas: (i) Requirements Engineering, with RSL; (ii) Test Engineering, with TSL (Testing Specification Language); and (iii) Project Management, with the PSL (Projects Specification Language). To improving the development of technical documents (eg, requirements specifications, test case specifications or project plans) more rigorous, ITLingo adopts a linguistic approach, which will also result in increased productivity through transformations and re-use of models, along with improvement in quality with semi-automatic validation. At an early stage, RSLingo, which accommodated ITLingo’s RE area, that the use of natural languages was the preferred and common way in specifying requirements, however these are conducive to the development of inconsistent ambiguous documents that are difficult to validate or automatically.

In order to solve quality problems in the production of SRS and at the same time provide a language that accommodates all interested parties, RSL is used, which helps mitigate some of the problems arising from the use of Natural Language for the production of requirements, where there is strict control and a right way to write SRS, and it also offers benefits in relation to software development [4].

RSL is a DSL with the main objective of improving the production of requirements in a more systematic, rigorous and consistent way using a Controlled Natural Language (CNL), which follows practical recommendations to improve the writing of the requirements. At the moment it is in an advanced state, having undergone several updates since its creation with the aim of developing a more extensive and consistent language (nowadays called “RSLingo’s RSL” or just “RSL”).

RSL provides several logically organized elements in two perspectives 1: level of abstraction and specific subjects. The levels of abstraction are: business levels, applications, software, and hardware. On the other hand, the specific subjects are: active structure (subjects), behavior (actions), passive structure (objects), requirements, tests, other subjects and relationships and sets.

This language is composed of an elaborate set of elements organized logically according to two perspectives: level of abstraction and specific concepts.

The elements of the RSL language are defined by linguistic standards and represented textually according to concrete linguistic styles. These
are organized according to two views: RSLOverview: the level of abstraction (Levels) and specific Concerns (Concerns). The levels of abstraction are: business levels, applications, software, and hardware. On the other hand, the specific concerns are: active structure (subjects), behavior (actions), passive structure (objects), requirements, tests, other subjects and relationships and sets.

Figure 1: RSL Overview[3].

From the practical point of view, any element can be used in any type of system, regardless of its level of abstraction. This means, for example, that you can use a DataEntity element at the Application or software levels, but also at the business or even hardware levels. However, the use of a DataEntity at the business level should be more general and incomplete (for example, without specifying data attributes) compared to its use at software levels, which should be more detailed (for example, including attributes data and foreign key references specification). In addition, while some elements (for example, Stakeholder, ActiveElement, Glossary-Term, Risk, or Vulnerability) are naturally applied to different types of systems, other elements are not so obviously applied (for example, UseCase and Actor are best applied only to applications or software) [6].

5. Technologies Used
Several technologies/frameworks were researched in this dissertation in order to know their strengths and weaknesses and choose the most suitable for this project. In the context of a computer engineer, the choice of technologies/frameworks should take into account many factors (for example, skills and team experience, context and project requirements, objectives, etc.), as the success of software development is influenced by this choice.

The technologies chosen for this dissertation were based on some advantages taking into account the main objective and context of this project and are listed next.

- Angular6 for client side development
- ASP.NET Core for server side development
- RSL for SRS specification
- Xtext/Xtend for SRS specification parsing and code generation

Regarding RSL, for this project, the focus will be only at the system level, and therefore the elements that will be used for the requirements specification are the following: Actor, DataEntity and DataEntityCluster, UseCase and StateMachine, the latter being for a job more advanced. The figure 2 shows the views and system-level elements of the RSL that will be used for this project. The blue area encompasses all the elements used for specifying requirements, the light blue area, which is the StateMachine area, represents an advanced work, and is not a priority for the initial development of this project due to time consumption and the difficulty involved. However, the blue area in general represents the constructs that are included for generating code for the web application, which will depend on what is specified in each of the constructs, which will be explained later in the report.

Figure 2: Elementos utilizados para especificação RSL neste projeto, adaptado de [3].

As already mentioned, all the elements and views represented in the figure 2, which will be used in this project, belong at the system level to the abstraction levels of the RSL. This is because, the functionalities of a system are represented mainly at this level. For example, the Actor element for the context of this project represents the actors that interact with the system,
which will usually result in the creation of a table of actors in the database and, which will allow login/registration and enable different users to have different roles (such as administrator, manager, user), and consequently different privileges. Likewise, the DataEntity construct can represent which items/entities will need to be created in the database. TheStateMachine construct will add "states" to the items/entities created by building the Dataentity. And finally, the UseCase construction will usually result in features for the system, such as: giving one or several actors the ability to search, create, update, etc. for an item/entity. While the business level, despite having an important role to enrich the description of a system, is not so relevant to this project since it does not represent system functionalities, and what we want is precisely functionalities for the web application.

5.1. System Overview
As for the process flow of RSL2WebApp for building a web application, it is subdivided into three tasks in the following way:

The first task, which is the only one that requires user participation, is SRS using the RSL language. At this point, the user must specify the application requirements using the RSL elements (Actors, StateMachines, DataEntities, UseCases). This specification in RSL must be valid, that is, without errors respecting the RSL grammar and keywords, so that the code generation for the web application with Xtend is possible.

The second step of the process corresponds to the source code artifacts produced by Xtend. There is a predefined model for generating code in RSL2WebApp, which encompasses all elements of the system level (Actors, DataEntities, etc.), the program will iterate over each item within the specification and generate source code, depending on specified by the user.

The third task is to create a web application project ready to run. The web application is compiled with Angular6 and ASP.NET Core for the client side and server respectively. The final product should be ready to run, although it is possible that the generated code does not meet all specifications written by the user and is partially incomplete, requiring that the code be completed manually.

In a very brief way, the generated code for each element of the RSL will follow the following model:

- **Actors** - creation of a table in the database with the different types of users (timers, managers, admins, operators)
- **DataEntities** - creation of a table in the database for all the entities involved and their machine status (invoices, products, etc.)
- **UseCases** - assign possible actions to one or more user types (create invoices, delete invoices, update invoices, browse / filter invoices, view)

6. RSL2WebApp Approach
RSL2WebApp, as mentioned before, consists of a tool that produces source code for web applications based solely on a SRS. The generated source code for the web application has as a startup template the public QuickApp project.

6.1. QuickApp Startup Project Template
QuickApp is a ASP.NET Core 2.2 / Angular 7 startup project template with complete login, user and role management.

This project includes a variety of features from the outset, mainly in the LogIn part, authentication, user management and role management, which will be useful for RSL2WebApp, since the generated application will also include them. The QuickApp project aims to aid and facilitate the start-up of web application development in a faster way "Quick Application Development", that has to be completed by RSL2WebApp, and consists mainly of the following characteristics.

- Template pages using Angular6 and TypeScript
- RESTful API Backend using ASP.NET Core 2.2 MVC Web API
- Database using Entity Framework Core
- Authentication based on OpenID Connect
- Theming using Bootstrap 4

These characteristics were another reason for choosing this startup project for the generated source code for the web application since it meets the requirements of this dissertation, which are to use Angular to the client side, ASP.NET Core to the backend together with the Entity Framework to assist in the construction of the database and that has a responsive design, which in this case is done with Bootstrap library.

6.2. RSL2WebApp source code generation
In this section explains how RSL2WebApp parses the SRS in RSL and generates the source code based on the SRS.

As previously mentioned, code generation is done with the Xtend language, within the Xtext project itself which contains the grammar for the RSL. In this way it is possible for the xtend class to generate the code, access to all entities defined in the specification by the user. In this case the DataEntities, DataEntitiesClusters, Actors and UseCases. The 3 image shows an example of

---

https://github.com/emonney/QuickApp
code generation where all the "Entity" defined in the grammar are being traversed and for each "Entity" a java file is generated, with the same name defined in RSL, within an "entities/" folder where the contents of that file will be generated by the compile method. In this dissertation the same idea is followed, where all the Entities defined in the RSL specification will be traversed and the files necessary for the web application will be generated.

```java
override void doGenerate(Resource me, FileNotFoundError sfa, IGeneratorContext context) {
    for (e : res.allContents.filter(filter(Entity))
        sfa.generateFile("entities/" + e.name + ".java", e.compile)
}
```

Figure 3: Método doGenerate Xtend

Regarding the backend side, since QuickApp already offers login and user management functionalities, in a summarized way, what remains to generate are the entities (tables) to be created in the database and their relationships, which will be defined in the DataEntities element within RSL. It is also necessary to generate the different roles (admin, manager, user) corresponding to the Actors element, associate them with the respective permissions that will be defined by the UseCases, and also create an API that allows operations such as add, delete, edit and read for each generated entity in the database, these methods are usually called CRUD operations (Create, Read, Update and Delete).

For the client side, in a brief way, it is necessary to generate two pages for each DataEntity to allow the user to perform the CRUD operations mentioned above, and also generate the services that allow to the client communicating and calling the API on the backend side with HTTP requests.

The image 4 shows the main components generated by RSL2WebApp and its architecture.

![Figure 4: RSL2WebApp Architecture](image)

7. Evaluation

The evaluation process will determine the performance of RSL2WebApp, where it is checked whether the entire system specification defined in RSL by the user has been correctly interpreted by RSL2WebApp, which includes all elements, attributes and keywords, and generated the respective code. Also evaluated is the percentage of generated code taking into account the manually developed web application that corresponds to the complete RSL specification. It will be possible, in general, to assess the level of effort and time, which on average can be spared in the development of SPA web applications, using RSL2WebApp.

It is also possible to determine the time and effort saved by taking into account the number of lines of code that are automatically generated in contrast to the lines of code produced by the user in the requirements specification in RSL.

Two pre-made RSL specifications were used for the evaluation. The first one corresponds to a simple virtual enterprise information system, called "Billing System" described in the next section), where it is verified that all the elements were generated correctly. RSL2WebApp should be able to generate code for web applications with any RSL specification, either simpler or more complex than the Billing System. The second system represents a fictitious virtual store called "MyStore".

7.1. Study Case

The first case study, called "Billing System" is a fictitious commercial information system. Its purpose is to show how to produce a specification using the RSL language in a real scenario. This fictitious system consists of a set of resources for managing customers, products and invoices. The specification of the billing system that are described belong to the level of the RSL system, but there is also a specification for the level of business that complements/enriches the specification. For this project, only the system level of the RSL is used for the generation of the code as mentioned previously, and therefore only the system level specifications for the evaluation will be used.

The second case study is the "MyStore" system which consists of an online store that will allow the user to make purchases by inserting products into a shopping cart. It aims to solidify and test RSL2WebApp better, exploring elements or keywords that were not so explored in the Billing System and also to verify that there are no dependencies of RSL2WebApp with the Billing System, since this was the web application developed manually and served as a guide for RSL2WebApp code generation.

7.2. Methodology

To be able to evaluate RSL2WebApp, to calculate the percentage of code generated, the following methodology will be adopted:
• Create a reference web application using Angular6 and ASP.NET Core manually for Billing System case study
• Specify the requirements of the web application manually created using RSL
• Generate the source code with Xtend for the web application using the RSL specification
• Compare the generated code with the original and determine the generation rate

7.3. Results
For the Billing System case study, the user-defined specification was correctly processed by RSL2WebApp and generated the desired code, with the exception of the keywords that are possible to define in RSL but are not supported by RSL2WebApp, as previously mentioned. RSL allows a specification with more elements besides the users by RSL2WebApp (Actor, DataEntity, DataEntityCluster and UseCase) and also allows to define properties that are not supported by RSL2WebApp and therefore, no code will be generated for these elements. Using only the elements and keywords supported by RSL2WebApp, after code generation, there is no need to add any line of code or make changes to the generated web application. The RSL2WebApp code generation models had as a reference the manually developed web application, which already took into account all the inherent limitations of keywords and types supported in the specification of RSL requirements by RSL2WebApp, namely the DataEntities attribute types and your primary and foreign keys. The generated web application is therefore fully generated when compared to the one developed manually, that is, the code generated for the web application together with the code already existing in the QuickApp reference application were 100% of the code previously developed to represent the system defined in the RSL specification.

The tests were also valid for the second system "MyStore".

8. Conclusion
In this work we discussed several issues related to web development that help us to better understand what a web application is, specifically a SPA, and what are the most popular technologies and frameworks for its development, both server and server side. client side. The advantages and disadvantages were also discussed, so that one can have a better view on the existing options, and choose the most appropriate depending on the technological requirements of each situation. Other subjects such as SRS have also been introduced and how important it is during project development, MDE and code generation. Finally, the proposed solution for the RSL2WebApp under development was presented, which included the technologies used for system specification, server-side development, client-side development, and code generation.

RSL2WebApp aims to facilitate and reduce the development time of responsive web applications, reducing their complexity, saving time and total cost when developing a web application. Generally, this can also reduce the risk of error introduction during development. Web applications generated by RSL2WebApp have as main characteristic the creation of entities in the database and enable the user to manipulate this data with create, remove and edit operations. In addition it allows that there are different users with different permissions regarding the possible operations to these entities. However the generated applications will always have the same style and structure in terms of presentation. It is still advantageous to use RSL2WebApp, since it is possible to later change the generated code, since it is well indented and organized, thus making updating or altering tasks easier. It was possible to conclude that, in fact, it is possible to save effort and time, as shown in chapter 7. It was also possible to conclude that the more detailed and enriched the RSL grammar, the more possibilities RSL2WebApp will have to make the web application more flexible and personalized. However this brings some disadvantages as it becomes more difficult to develop the specification of system requirements in RSL. One of these disadvantages is that RSL aims to be a simple and easy-to-use language, but if it is enriched with more keywords to allow the generated application to be more flexible, it can become more difficult to use and complex. This was against the main objective of this dissertation, which was to reduce the complexity and time to develop web applications and also to enable non-technical people with little experience in code development to be able to use RSL2WebApp.

References


