A Requirements Elicitation Tool for Document Migration of Enterprise Content Management Systems

Kadir Yozgyur
Abstract

A Requirements Elicitation Tool for Document Migration of Enterprise Content Management Systems

Kadir Yozgyur

Requirements Engineering (RE) is the area of Software Engineering that deals with finding, analysing, and structuring requirements from stakeholders of a project. Requirements Elicitation (RL) constitutes arguably the most crucial activities of Requirements Engineering. It is concerned with collecting, searching for, and elaborating requirements from stakeholders. The aim of this work is to provide a RL tool to improve the quality of requirements by enabling increased user participation, easing the effects of time constraints, and providing a common recording medium. The subject area selected for this thesis implementation is the elicitation for migrating physical documents into an ECM system that is being built. Requirements Interchange Format (ReqIF) is used as the underlying data model to rely on a solid standard. AJAX technologies and several JavaScript libraries are used together to create a smooth User Interface.
Contents

Abstract i

Contents ii

List of Figures iv

List of Tables v

Abbreviations vi

1 Introduction 1
  1.1 A Definition: Requirements Elicitation 2
  1.2 Role of Requirements Elicitation 3
  1.3 Research Focus 4
    1.3.1 User Involvement 4
    1.3.2 Application Area 5

2 Literature Review 6
  2.1 Introduction 6
    2.1.1 Requirements Elicitation Techniques 6
    2.1.2 Requirement Elicitation Methodologies 8

3 Project Design 10
  3.1 Project Domain 10
  3.2 Objectives 11
    3.2.1 Improving Quality of Requirements 11
    3.2.2 Time Factors 13
    3.2.3 Providing a Recording and Observation Medium 15
  3.3 Design 15
    3.3.1 Process Flow 16
    3.3.2 Meeting Structure and Tool Workflow 17
  3.4 Technical Choices 18
    3.4.1 Model: ReqIF Standard & Eclipse Requirement Modeling Framework 19
      3.4.1.1 Eclipse Requirement Modeling Framework GUI: ProR 20
List of Figures

3.1 Overall process flow with the proposed tool ...................... 16
3.2 An example meeting workflow with the proposed tool ............ 17
3.3 GUI grids mock-up .............................................. 22
3.4 Requirement links mock-up .................................... 23
3.5 Diagram mock-up ................................................... 24
4.1 Overall architecture ............................................... 26
4.2 UML class diagram for the data model .......................... 37
4.3 UML class diagram for the server-side implementation .......... 38
4.4 UML class diagram for the client-side implementation ........ 39
4.5 A screenshot of the finished implementation .................. 40
4.6 A screenshot of a generated diagram ............................ 41
5.1 Scenario 1 - Creation of two documents, two requirements, and a relation 47
5.2 Scenario 2 - Preserving consistency during deletion operations .... 47
List of Tables

5.1 Identified Risk Classes ............................................ 43
5.2 Risk Priority Ranges ............................................. 44
5.3 Designed Test Suites ............................................. 44
5.4 Test Cases for Document Form .................................. 45
5.5 Test Cases for Field Requirement Form ...................... 45
5.6 Test Cases for Other Requirement Form .................... 46
5.7 Test Cases for Relation Form .................................. 46
5.8 Test Results for Document Form (Table 5.4) ............... 48
5.9 Test Results for Relation Form (Table 5.7) ................. 48
5.10 Test Results for Field Requirement Form (Table 5.5) ... 49
5.11 Test Results for Other Requirement Form (Table 5.6) ... 49
5.12 Test results of Scenario 1 (Figure 5.1) ..................... 51
5.13 Test results of Scenario 2 (Figure 5.2) ..................... 51
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE</td>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>RL</td>
<td>Requirements ELicitation</td>
</tr>
<tr>
<td>ReqIF</td>
<td>Requirements Interchange Format</td>
</tr>
<tr>
<td>RMF</td>
<td>Requirements Modelling Framework</td>
</tr>
<tr>
<td>DMS</td>
<td>Document Management System</td>
</tr>
<tr>
<td>ECM</td>
<td>Enterprise Content Management</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>JAD</td>
<td>Joint Application Design</td>
</tr>
<tr>
<td>ORDIT</td>
<td>Organizational Requirements Definition for Information Technology</td>
</tr>
<tr>
<td>IBIS</td>
<td>Issue-Based Information System</td>
</tr>
<tr>
<td>UCSD</td>
<td>User-Centered System Design</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>MVVM</td>
<td>Model-View-View Model</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript And XML</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A software system that meets the expectations of the clients and users should be developed on the basis of correct and appropriate requirements [1]. This statement is almost common sense because it can be generalized into any kind of solution in any kind of domain. It is only logical that a solution is doomed to fail if it does not reach the desired objective. Requirements Engineering (RE) is the area of Software Engineering concerned with finding and assuring this objective. Requirements Elicitation is the stage of RE where the required information is extracted and collected from the client stakeholder.

A requirement is a capability that this system must have in order to solve a problem or achieve an objective within a specific domain [2]. Studies show that poor requirements are one the greatest factors that lead to project failure [3, 4]. An important problem in RE is the quality of elicited requirements. It is beneficial, in this sense, to include as many stakeholders as possible. However, the time costs associated with elicitation activities are important constraints to consider. Another important problem is the recording of elicited requirements.

In this thesis, an online tool for requirements elicitation is proposed that is geared towards nonengineer stakeholders’ use. The tool should address these issues in requirements elicitation by providing a user-friendly interface that provides support and feedback in the form of diagrams. The core model to be used by the tool is the Object Management Group (OMG) standard, Requirements Interchange Format (ReqIF) [5]. Particularly, the Eclipse-based open-source platform, Requirements Modeling Framework (RMF) [6], has ReqIF as its data model and serves as the base for the proposed tool.
1.1 A Definition: Requirements Elicitation

RE is a key problem in the development of complex software systems [7]:

The hardest single part of building a software system is deciding what to build... No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

Every project has an ultimate goal that corresponds to the question of ”what” [8]. This goal is not concerned with ”how” to realize a solution but rather names the task that is to be done. In parallel, RE is not concerned with the implementation of a project solution but rather identifying, analyzing, and specifying the objectives that are aimed.

Zave [9] describes RE as:

Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems.[...]

RE consists of different activity-stages in order to reach concrete requirements. The naming of said activities is not universally agreed upon while their execution also varies with different types of systems being designed [10]. Below is a list of these activities.

1. Requirements elicitation
2. Requirements analysis and negotiation
3. Requirements specification
4. System modeling
5. Requirements validation
6. Requirements management

Requirements elicitation is the process of gathering requirements from the stakeholders. The word ”gathering” does not really cover the whole meaning implied by ”elicitation”. Elicitation also includes the meaning of mining for information to compile them into logical constructs. Requirements analysis is the name of activity that is done to analyze requirements. Requirements specification activity aims to produce a specification document from requirements. System modeling is concerned with deriving models for the system from the requirements. Requirements validation is the activity of validating the correctness of requirements and management is concerned with managing evolving requirements.
The scope of these activities are the subject of debate among practitioners of RE. A natural reason for this is that every application of these practices in different systems domains differs from one another. There is no universally accepted, perfect RE process that would give good results in whatever system it is being applied on. So the attitude towards requirements activities is to focus on succeeding rather than always trying to follow a concrete course of action. As this is the case, this thesis project will focus on improving Requirements Elicitation (RL), and hence, a definition of and what is exactly meant by RL is needed to be defined.

RE activities usually do not take place sequentially but rather intertwined. So the scope of each named individual activity can become blurry. In this thesis, RL is considered to be the focused activity because the elicitation activity has the most focus. With this said, the focus of this thesis also includes some elements of other activities. One of such elements is to provide analysis possibility for requirements. This is inherently a part of the elicitation process. Eliciting requirements includes analyzing already elicited requirements to make them better or find other requirements etc. Management of requirements is also made possibly with the thesis implementation. The tool is designed in a way that the output is already in a very close form to a requirements specification document. So, the meaning implied by RL in this thesis is, in many places, very comprehensive and usually could very well refer to the whole RE itself.

1.2 Role of Requirements Elicitation

Researches have identified the major cause behind software project failures to be poor RE [4]. Requirement elicitation is arguably the most important part of the whole RE process. It is concerned primarily with the communication between client stakeholders and engineer stakeholders. A successful level of elicitation early on in the project is important in order to create a solid mutual understanding so as to not leave some project goals out. These left out goals produce a lot of dissatisfaction for the final project and may imply a lot of work, and cost, to implement later on. Another kind of failure is misunderstandings of requirements. Misinterpreting and wrongly implementing requirements produce just as bad results as not implementing them at all.

There are many elicitation techniques and also methodologies that propose complete roadmaps using a combination of different techniques and tools. Some of these techniques are interviews, workshops, observational and documentation studies etc. Each technique comes with its varying associated costs. For example, elicitation through observation of work environment takes a significant amount of time while interviews take less time in comparison. Each technique has its particular effectiveness in particular situations. Again, comparing interviews to observation, interviews are better at the beginning of elicitation where verbal communication produces a big amount of initial data whereas observation would be hard to conduct because the elicitation specialist would not be so aware of what to observe. In contrast, the amount of information
elicited through interviews decreases as elicitation advances and more valuable information could be achieved through observation. Understanding which technique suits which situation, what are the costs and benefits, is a virtue that RE experts acquire. An expert, to achieve good result from elicitation, should be able to decide what technique to use in which situation to maximize elicitation output quality.

### 1.3 Research Focus

Communication of ideas is a very important issue which is, compared to social sciences, unfortunately not so well structured by software engineering. When technical aspect of computer science is considered, it does not seem to be intricately related to anything of social sciences. However, possibly the most important factors in software engineering come from the social aspects in which it is performed. In this way, requirements elicitation possibly carries the gravest importance. It is directly concerned with communication between engineers and their clients.

Among many problems present in a typical elicitation process, quality of requirements seems to be the ultimate consideration. To reach a good level of quality in elicited requirements is the most important goal of requirements elicitation. However, the elicitation process is intertwined with concerns of communication in an organizational environment. This fact brings it many parameters which affect the overall performance of elicitation. This thesis project is concerned with some of these parameters alongside quality concerns: time costs of elicitation and organization of elicited requirements. Time costs affect the performance of elicitation by prohibiting the amount of elicitation activities that could be performed or the number of people that could be included. Also, information gathered from elicitation activities pose an organizational problem. These factors, together with the main quality concern, is taken as the focus of this research and to remedy these problems, a tool is hereby proposed.

### 1.3.1 User Involvement

It is important for the elicitation process to include as many stakeholders as possible to ensure an adequate coverage of requirements. In its most basic level, RL is about communication and mutual understanding. When the fact that every stakeholder person has their unique perspective of the system is considered, the importance of communication and understanding of each and every perspective becomes evident. However, the greatest obstacle in the way of achieving this is the time constraints and costs associated with elicitation activities. In order to not stall the company operations, employees who are needed to take role in the activities will need to continue their regular work activities alongside elicitation activities. This reduces the time that can be spared for elicitation activities and more importantly puts very limiting constraints on the scheduling of said activities.
Another setback is the cost of spent time in elicitation by the RE engineers. The aforementioned difficulties in time dedication and scheduling of the stakeholders force RE specialists to dedicate more time to elicitation. This reflects as more time cost to the software company. Following this, the software company may not be so willing to compensate these increasing time costs which will lead to poor requirements elicitation.

Especially in large-scale and distributed software projects, it is unfeasible to organize personal meetings on a regular basis [8]. However, it is arguable that meetings are still not feasible enough in smaller and localized settings. A popular solution in requirements elicitation in large-scale software projects is wiki-based forums [8] but they are susceptible to information overload, redundancy, incompleteness of information, and diverging opinions of stakeholders.

1.3.2 Application Area

In today’s world, many organizations are facing the challenge of adapting to the fast pace of modern, technological society. The many types of organizations (e.g. institutions, companies, non-profit, etc.) that share this common problem. Throughout this thesis, these different types of organizations will be referred to as ‘organizations’. In today’s technological world, old organizations need to rapidly adapt themselves to the new fast pace of things. This adoption of the digital world is essential in survival of many organizations.

Enterprise Content Management systems serve as the main pillar of this transition. They are descendants of Document Management Systems (DMS) which were used to govern documents by capturing images. ECMs have succeeded DMSs with more advanced technology and comprehensive scope. However, the document management necessity still remains and constitutes an important part of the transition of old organizations into the digital world. The focus of this thesis project is to aid RL for software solutions that aim to perform this migration to digital document management systems. A tool implementation is presented to be used during RL activities.
Chapter 2

Literature Review

2.1 Introduction

As reflected in this quote from the year 1987 [11], requirement elicitation did not receive much attention from the software engineering research community in the past:

From the survey, it was learned that requirements analysis, in particular requirements elicitation, is a hard task, and that it is carefully avoided by most of the software engineering researchers. We believe that most researchers avoid dealing with elicitation of requirements, because it is an area where one has to deal with informality, incompleteness and inconsistency. Instead, research labeled as dealing with requirements, usually deals with specification, and that is the main reason for the lack of agreement on the definitions of requirements analysis and specification.

In the recent years, there have been increased research interest in this area. The techniques are better investigated and complex methodologies with precise directions have been formed. A selection of these techniques and methodologies will be presented in this chapter.

2.1.1 Requirements Elicitation Techniques

There are many techniques and methodologies that address requirements elicitation problems [12–17]. They are classified in different ways in different sources. However, most of the classified techniques themselves are well established and the same among these different classifications and so the classification in [17] is chosen to be followed in this paper.
First type of techniques is called *Conversational Methods*. This class of techniques includes interviews, workshops/focus groups, and brainstorming. Since they use natural language, unlike specialized tools, they do not require technology support. All of these techniques rely on verbal communication as the primary way of elicitation. Out of these techniques, interviews are the naturally easiest, most cost-effective, and hence most commonly used technique. Reference [18] states that:

Even a few hours of interviews can be very valuable, even when conducted by engineers who have had only brief training in interviewing and user needs analysis.

Workshops are a group technique where people gather together in a meeting to investigate and create ideas. It usually has a predefined topic to be discussed, i.e. a part of the system. The specialist is expected to plan the details of discussion such as what should be inquired from participants, how much time should be dedicated to each part of the topic, etc.

Brainstorming is another group technique where ideas are voiced by each member and collected. It helps to foster a creative environment that could be beneficial during the early stages of development. During a brainstorming session, there are two phases [19]-the generation phase, where ideas are collected, and the evaluation phase, where the collected ideas are discussed. The generation phase consists only of the ideas being shared and collected in a nonselective manner. Brainstorming leads to a better problem understanding and a feeling of common ownership of the result.

Conversational Methods are the most commonly used techniques. They are similar to each other in nature and so one can easily incorporate elements from another. For example, a workshop can include a brainstorming session or can be thought of as a group of interviews taking place concurrently. While they are the most popular techniques, they are labor intensive [17]: they require much time investment, the recording and analysis of the meetings become arduous, and the scheduling of meetings is a challenge in itself. The organization of the information collected through these methods is a neglected issue [20] which is meant to be addressed in this thesis.

*Observational Methods* rely on observing the human activities at the working environment. People rely on their experience and intuition when performing tasks. These kinds of details may not be easy to articulate in face-to-face elicitation activities. Some of the Observational Methods are social analysis, ethnographic study, and protocol analysis. These techniques are carried out by the RE specialist through observing and analyzing the working environment to find out implicit details that stakeholders might not readily see the need to voice.

*Analytic Methods* provide ways to explore the existing documentation or knowledge and acquire requirements from a series of deductions [17]. By exploring documentation on regulations and company structures, some valuable information could be gathered.
Documentation studies/content analysis technique consists of this exploration. Requirements reuse technique is to try and reuse requirements from, say, another similar project; analyzing its requirements specification and extracting what might be relevant to the project at hand as well. Techniques such as laddering, card sorting, and repertory grid all aim at extracting expert’s knowledge. These techniques are also called Cognitive Techniques [15].

The last type of techniques are Synthetic Methods. A single technique listed above will most likely fail to address the whole requirements elicitation process because each technique has its limitations. It is better to use some techniques together to reach more comprehensive results; to perform adequate requirements elicitation. Synthetic methods are such combinations that implement a coherent system in themselves rather than merely being a combination of methods. In practice, the practice of elicitation Some of such methods are scenarios, storyboards, prototyping, JAD/RAD workshops, and contextual inquiry. These methods have overlapping characteristics with methodologies.

2.1.2 Requirement Elicitation Methodologies

Requirement elicitation methodologies constitute complete frameworks of guidelines. They build on top of traditional techniques with intricate details that should be followed when applying a particular technique. They aim to provide more clear instructions to people involved in the elicitation process. The difference between terms methodology and technique/method is described as follows [21]:

It is the essence of a methodology- as opposed to a method, or technique- that it offers a set of guidelines or principles which in any specific instance can be tailored both to the characteristics of the situation in which it is to be applied and to the people using the approach... Such is the variety of human problem situations that no would-be problem solving approach could be reduced to a standard formula and still manage to engage with the richness of particular situations.

One such methodology is Joint Application Designing (JAD) [22]. JAD focuses on improving the group process and has been used by IBM since the late 1970s [20]. Some of the advantages of JAD is the promotion of cooperation and understanding between stakeholders. JAD uses structured meeting procedures, some protocols, and visual aids. A recognized problem with JAD is that all the ideas are collected by a facilitator/recorder. This problem is prevalent in other techniques as well. The recorder may, knowingly or unknowingly, impose their own perspective onto the recorded information while collecting and organizing it.

Some methodologies also define organizational and context analysis as an explicit first step in elicitation, to be followed by other information gathering activities [20]. These
methods emphasize the importance of defining the organizations’s objectives and con-
straints against those objectives. The ORDIT (“Organizational Requirements Definition
for Information Technology) methodology, the product of ESPRIT II project, empha-
sizes the definition of organizational requirements as well [23]. The ORDIT method-
ology recognizes that users work in an organization, have organizational goals, and are
subject to organizational constraints.

The IBIS methodology (Issue-Based Information System) focuses on structuring ra-
tionale underlying requirements to be organized and tracked [24, 25]. IBIS has some
shortcomings that are expressed: IBIS is vulnerable to scheduling pressure whereby
documentation is abandoned in favor of getting code written and system working [25].
This scheduling pressure actually applies to documentation of all other elicitation tech-
niques as well [20].
Chapter 3

Project Design

In this chapter, the thesis project design is presented. First, the type of projects that the design will focus on eliciting requirements for is explained in detail. The project domain for the thesis implementation is the digital document information capabilities of Enterprise Content Management systems. In the second section, objectives of the thesis will be presented. These objectives are improving the overall quality of requirements, reducing time cost and scheduling constraints, and providing a recording and working platform for requirements.

3.1 Project Domain

Many organizations today are facing the difficulty of catching up with the ever-increasing speed of the modern digital world. As the technology continues to develop, more and more businesses depend on electronic means to continue their existence in the competitive world. Even though this trend is globally engulfing every aspect of our lives, and hence, every area of business, there are still many organizations trying to make their transition into the digital world. The survival of many small to middle sized businesses depend on their success of making this transition.

To make this transition, organizations are looking for solutions to enable their operations to go faster with better and easier traceability. Of course, the logical way to address this issue is to embrace the changing mechanics and adopt the technology as the workhorse of operations. The typical track that is followed on this account is to migrate from management of old paper-based documentation to digital systems. The selected project domain for this thesis covers these digital document management systems. These systems are imagined to be used in a mid-scale organization (e.g. around 50-200 employees) to manage their information digitally.
Document Management Systems (DMSs) emerged in the 1980s as software systems that managed paper-based documents. These systems were used to capture images of paper documents and store these digital copies for easy access. As the technology advanced, functionality like text-recognition, indexing for easy retrieval, etc. are added. Today, DMSs are succeeded by the more comprehensive Enterprise Content Management (ECM) systems. Enterprise Content Management is the integration and utilization of one or more technologies, tools, and methods to capture, manage, store, preserve, and deliver content across an enterprise [26]. ECMs can be designed to handle all levels of organizational operations such as content management, organizational workflows, business processes as well as other peripheral services such as security and easy document retrieval.

The focus of this thesis implementation is on ECM systems for small-to-medium-scale organizations. The implemented tool is designed to aid requirements elicitation of such systems. Designing a absolutely comprehensive requirements elicitation tool for ECMs is obviously an overly arduous task to be undertaken in a master’s thesis. Among many different functionality usually offered within ECMs, only a particular part is selected as a model feature for the tool, that is, eliciting requirements for document content management. The informal types of requirements related to document content management that are being addressed in this thesis are the actual content/fields of documents, field relations between other documents, and possibly workflow or other kinds of requirements that are not directly content-based.

3.2 Objectives

There are many established techniques for requirements elicitation as listed in Chapter 2. The main objective of this thesis is to approach some common problems in these techniques and devise solutions in the specific domain described in the previous section; ECM systems. The standpoint of the tool implemented with this thesis is increasing the direct stakeholder input to improve the ‘quality’ of the elicited requirements and to reduce the effect of time constraints on the project. The other objective is to provide RE specialists and other stakeholders a common ground for recording (and modeling) requirements.

3.2.1 Improving Quality of Requirements

Tacit knowledge is the type of knowledge that is implied rather than explicitly stated. Since requirements elicitation is inherently a transfer of knowledge, the recognition and articulation of tacit knowledge poses an important challenge to overcome. Reference [27] states that the importance of tacit knowledge in RE is widely acknowledged because the inability to extract and understand tacit knowledge can lead to project failure-
through poor requirements elicitation. According to the Tacit Knowledge Framework (TKF) presented in [27], there are four classes of knowledge when dealing with requirements [13]:

- **Known-knowns**: expressible, articulated, and relevant.
- **Known-unknowns**: not expressible or articulated, but accessible and potentially relevant.
- **Unknown-knowns**: potentially accessible but not articulated.
- **Unknown-unknowns**: not expressible, articulated or accessible but still potentially relevant.

In order to have a coherent flow of explanation, the second and third classes will be explained followed by the first and last. The first word in the names of these classes represents the specialist’s understanding of the knowledge while the second word corresponds to the stakeholder’s. So the second class, known-unknowns, are knowledge that the RE specialist knows that they exist but the stakeholder is unaware of this knowledge. An example for this type of knowledge can be the expected traffic volume of a web-based project. It is relevant because it is a factor that needs to be addressed in order to deliver a project that can serve the expected traffic. The specialist will naturally be aware of factors like this and will make inquiries about them to the stakeholder. If the RE specialist does not have specific information about the project domain outside the project technicalities, this type of knowledge will not be very prominent but it can occur more as the specialist gets accustomed to the domain through the elicitation activities. It can only exist if the specialist is one step ahead, so to speak, of the stakeholder.

Known-unknowns do not pose a problem that is immediately addressed by this thesis. The formation and articulation of such knowledge depends more on the domain experience, as well as the overall professional experience, of the specialist. If the specialist is competent enough to formulate such knowledge and to pursue it, the elicitation problems here will be mostly addressed. The help provided here by the proposed thesis tool could be that the existence of an organized recording of the requirements could inspire the specialist to look for missing links and more information, hence, leading to the formation and eventually articulation of known-unknowns.

Is there a case where the stakeholder is ‘one step ahead’ of the specialist? We argue that the third class of TKF could partially match with this scenario. In this class, the knowledge is already held by the stakeholder, it is relevant to the project at hand, nevertheless the specialist is unaware of this knowledge. Since the specialist is unaware, he/she cannot assess the importance of this knowledge. This information, if not elicited, might lower the final satisfaction of the customers or be so crucial to even render the project completely crippled. To remedy this, we can be inspired by the process that creates previous class of knowledge, known-unknowns. As previously mentioned, this type of
knowledge is where the specialist 'knows' to inquire about some knowledge that is 'unknown' to the stakeholder. For unknown-knowns, the situation is somewhat opposite of this but with one crucial difference: the stakeholder does not know that the information he/she holds is potentially relevant so the stakeholder cannot reach the conclusion that the information he/she holds should be articulated and presented to the specialist.

The class of unknown-knowns is addressed by the tool’s focus on stakeholder contribution. The aim of this thesis is to shift some of the responsibility to stakeholders by informing them more deeply about the elicitation process and providing them with a usable tool. By using the tool and being more directly involved in the elicitation process the stakeholders are hoped to be more aware of the nature of elicitation process. The result of this is that, like the inquiry that specialist does for known-unknowns, stakeholders will ultimately remedy unknown-knowns by learning to voice them.

The known-knowns class constitutes knowledge that is known by both parties and hence can be considered already elicited. However, this does not necessarily mean that these known-known requirements are recorded and formed into concrete requirements specification, but rather that the elicitation/communication part is accomplished. The proposed tool will provide a ready recording ground for this class of requirements knowledge.

The last class of knowledge is the unknown-unknowns. Here both the specialist and the stakeholder do not have the knowledge. Exploration of possible missing information links is necessary to find this type of knowledge. Such activities could be brainstorming and analysis of already elicited requirements. Reference [13] goes deep into exploring the performance of various techniques on unknown-unknowns. On this account, the proposed tool will lie parallel to existing techniques as it does not introduce a new technique but rather a recording ground and a somewhat more extensive perspective.

To summarize, according to TKF classification, requirements in the first class pose no problem since they are already known and documented. RE specialist knows the requirements belonging to the second class of requirements. These requirements are needed to be asked for from the stakeholders. The third and fourth classes of requirements are where the proposed tool could be helpful with. With the third class of requirements, the stakeholders have the knowledge but the RE specialist is unaware of the need to elicit this knowledge. The tool will involve stakeholders more directly into the elicitation process to enable recognition of this type of knowledge. The fourth class of requirements is where both the RE specialist and other stakeholders are unaware.

### 3.2.2 Time Factors

The quality of requirements depends on the convergence of the other three classes of knowledge unto the first one. A difficulty factor in achieving this convergence, that is not stressed enough in RE literature, seems to be the end-user/stakeholder involvement in the elicitation process. User-Centered System Design (UCSD) discipline, for
example, concentrates on the end-users throughout the software design process. Active user involvement in the design process (for the purposes of this paper, requirements elicitation) is one of the key factors in UCSD [28]. Also, [20] states that

 [...] Effective requirements development depends upon the participation of everyone affected by a proposed system. Each participant involved in the requirements development process has a different view of the target system, and describing any participant’s view of the system or environment will tend to improve the overall understanding of that system.

The message here is that each participant has their unique perspective of the system and could provide valuable input to the elicitation process. Another point is, if everyone who is affected by the system is not involved in the elicitation process, the resulting project might not be accepted as satisfactory by everyone. However, the techniques described in Chapter 2 are all subject to time constraints and scheduling issues. This results in stakeholders being represented by a selected few instead of reaching all stakeholders themselves.

Except for the Observational Methods, the existing elicitation techniques are applied through meetings between stakeholders and RE specialists. Especially in large-scale and distributed software projects, it is unfeasible to organize meetings on a regular basis [8]. However, even in smaller and localized settings, it is arguable that meetings are still not feasible enough to conduct to properly realize the requirement elicitation process. While the Conversational Methods are the most commonly used elicitation techniques, they are heavily subjected to time constraints imposed by the existence of a working company. The elicitation process for software systems rarely happens in an isolated environment where stakeholders can dedicate 100% of their time to it. Employees who are needed to take role in the elicitation activities also need to continue their regular work activities at the same time. This reduces the time that can be spared for elicitation activities and puts very limiting constraints on the scheduling of said activities. The stakeholder may be unwilling to compensate these increasing time costs which will lead to poor requirements elicitation and which, in turn, might lead to overall project failure.

On top of the time costs, there is the problem of scheduling. In an environment where usual operations are taking place parallel to elicitation activities, it poses a problem to schedule meetings between people from different departments, or even within a single department, with different schedules. Together with the time costs, this increases the impact of time constraints, hence, make it more difficult to include as many stakeholders as possible. This naturally results in many meetings with different stakeholders spanning a wide time-frame which introduces a substantial amount of organizational overhead for recording requirements. To remedy the organizational problems associated with scheduling problems, the proposed tool offers flexibility in time management. It provides a common ground where progress in the form of requirement 'records' are kept, which is elaborated on the next subsection.
3.2.3 Providing a Recording and Observation Medium

Simple recording methods such as text, lists, sketches, etc. appear to be the accepted practice in recording requirements knowledge. These recording methods all prove to be unfeasible or unhelpful because of the essential difficulty in organizing and using the produced records. Records produced through these methods are not naturally tied with the requirements that they lead to, hence tracing between requirements and records taken in interviews, workshops, etc. gets ever harder as the elicitation process goes further and record volume increases accordingly. The proposed tool aims to remedy the recording problems as well as previously presented problems by providing an online platform that is easily reachable anytime and open for simultaneous modifications of different parts.

The connected objective that will be reached through recording capabilities is to provide an observation ground for requirements. As the records are being produced, they will also constitute a modeling ground for requirements as well. This will bring together collection, analysis and review of the requirements into one common place.

3.3 Design

This thesis aims to propose a web-based tool to reach the objectives presented in the previous section. The objectives are focused on some aspects of the requirements elicitation in the presented domain (ECM). Increased direct stakeholder contribution to the requirements elicitation process could improve the quality and coverage of elicited requirements. Being web-based, tool could constitute a solution to the time scheduling problem because of the accessibility it presents. By providing a common environment for both RE specialists and stakeholders alike, the tool addresses the objective of recording and analyzing requirements in a traceable fashion.

The objectives listed in the previous section follow each other. The encouragement of generation of unknown-knowns could benefit from an increased stakeholder involvement which is obscured by time and scheduling constraints present in a company/organization setting. A common platform where the records could be kept and improved over a span of multiple meetings could aid in subsiding the time and scheduling constraints. While helping time problems, a take on providing analysis capabilities is also of benefit. The proposed analysis capabilities in this thesis are diagram generation following from recorded requirements. The recording of information communicated in meetings in form of simple notes and sketches makes the organization of this information difficult hence resulting in the specialist not being able to use it effectively in formulating requirements.

To summarize the design aims, the tool should:

- Be easy to use
• Provide diagrams for inspection of the requirements

To reach the objective of involving stakeholders, the tool should be easy to learn. One thing that should not be forgotten is stakeholders basically do not share the same profession as RE specialist engineers. They will not have the same background as the software engineers that enables them to look at things and immediately construct a model in their minds. What should be remembered is this difference between the stakeholders and engineers. The tool does not expect stakeholders to learn ‘the engineer way’ but rather try to construct a communication channel which enables them to provide more input that might otherwise be missed (unknown-knowns).

3.3.1 Process Flow

An example flow to introduce and use the designed tool is presented in Figure 3.1. The first process is the introduction of the tool to the stakeholders. How to use the tool, the capabilities it provides, the benefits to be gained should be explained in this preliminary meeting. The stakeholders should be made aware of the whole process that is introduced by this tool. This step is crucial in the success of the tool because without establishing a proper understanding and mutual acceptance, it would be impossible to utilize it up to an acceptable level.

Next processes concern the thesis tool directly. There are supervised/unsupervised workshops/meetings and ‘free input’ outside these workshops. The supervision for the meetings are to be done by RE specialists. The structure of these workshops/meetings is up to the RE specialist’s preferred methodology. The tool is responsible only for providing a working area for these activities and so, for the sake of simplicity, they will be referred to as simply ‘meetings’ throughout the thesis. Unsupervised workshops will bring in much time cost reduction on the part of the software company undertaking the project. However, to be sure that the process is followed correctly, the output of these meetings should always be checked and validated by RE specialists in supervised workshop meetings.

![Figure 3.1: Overall process flow with the proposed tool](image-url)
The 'Free Input' process is optional. Outside these meetings, stakeholders could still be able to use the tool. A popular solution in requirements elicitation in large-scale software projects is wiki-based forums but they are susceptible to information overload, redundancy, incompleteness of information, and diverging opinions of stakeholders [8]. The thesis tool implementation could provide some functionality similar to those of wiki-based systems, such as change requests and comments. If implemented and put to use, this functionality should only be used as intermediary 'note-taking' in between the usual meetings sessions. While this collaborative note-taking functionality could be beneficial in recording agenda for future discussions, if unchecked, could bring in the usual problems associated with wiki-based forums and create confusion instead of helping the process.

The other RE activities such as validating elicited requirements and creating specifications are outside the scope of the thesis; they are represented by the last process on Figure 3.1. As proposed in [29], the whole RE process could be modeled as an iterative process. While the dashed line symbolizes the end of scope, the double-arrow through this line represents the possible iterative nature of the RE process. The recommended practice with the tool is to use it in this frame of mind; as a backbone to collecting and recording requirements.

### 3.3.2 Meeting Structure and Tool Workflow

A meeting workflow is presented in Figure 3.2. The workflow is iterative in its nature. It starts with manual requirements input to the tool. The foremost source of information for this input is extracted from the physical, paper-based documents to include in the final ECM system. The other source of information is the feedback from previous meetings, or the notes taken outside meetings. As the input is entered, it is recorded by the tool as data to be kept. Then from this data, some feedback in the form of diagrams are generated. These diagrams and entered data is then investigated which would lead to the whole cycle being repeated over and over again until the elicited requirements are deemed to be ready by the RE specialist.

![Figure 3.2: An example meeting workflow with the proposed tool](image-url)
The thesis tool primarily consists of a web-based solution for capturing requirements. The design of this thesis project focuses on the chosen application domain, that is, document management of ECM systems. As seen in Figure 3.3, the designed GUI will consist of data grids that are related to each other. First grid will list the documents with their names and descriptions. Each physical document will have a corresponding row on this grid. When a document row is selected on this grid, two related grids will be shown that list ‘field requirements’ and ‘other requirements’ for the selected document. Field requirements will constitute the actual fields on a document. For example, if the document in question is a passport, some example fields would be passport number, passport holder’s name and surname, date and place of birth, etc. The second grid is where other requirements that are not fields will be kept. For example, if the passport is a passport from a country that uses a different alphabet, should the name be recorded in both alphabets? Or are there any other documents that are needed to be present before a passport can be issued? These type of requirements that are not explicitly inside the document in question should be kept in this second grid. As it can be inferred from the mock-up, the idea is to provide a simple GUI.

The entered requirements may have relations among themselves. This idea is inspired by the Requirements Interchange Format (which will be discussed in detail later on). The idea is creating links between two requirements that defines their relation to each other. These links could be between requirements of different document or within the same document. For example, let say we have two documents with issue date fields. If there is a relation between these two dates such as one cannot be after another, this relation will be defined as a link. As seen in Figure 3.4, first a requirement of a document is selected on its grid. When selected, the links associated with the selected requirement will be shown to the user and on this grid, links will be added, updated, or deleted. The description fields will show which requirement each link is associated with.

In the third mock-up, Figure 3.5, example diagrams are seen. The field and other requirements of each document is listed in separate boxes together with the links connecting individual requirements. Diagram generation will be tied to a button. Whenever the diagram is needed to be investigated, it will be created with the present state of requirements.

3.4 Technical Choices

In this section, selected technologies for implementing the thesis tool will be presented. The whole web application architecture is analogous to a Model-View-Controller (MVC) architecture. The Model in this analogy is built on the ReqIF standard while the View is implemented with JavaScript through the use of open-source libraries KnockoutJS, JointJS, and jQuery. Lastly, the Controller part consists of Java servlets that handle request from the client-side and govern the data model.
3.4.1 Model: ReqIF Standard & Eclipse Requirement Modeling Framework

Requirements Interchange Format (ReqIF) is an OMG standard that is designed to provide a reliable and common way to record requirements [5]. ReqIF was created in 2004 by the Hersteller Initiative Software, a body of the German automotive industry that oversees vendor-independent collaboration [8]. The idea is inspired by the need to communicate requirements between car manufacturers and their suppliers. Before ReqIF was designed, this exchange was made through common tools such as Word and Excel and their file formats. ReqIF was created as a file format to replace these technologies to provide a standardized exchange capability.

Even though ReqIF was initially created as a file-based exchange format, inventors argue it can be more than that [8]. They liken the ReqIF’s relation to requirements with UML’s relation to model-driven software development. After the specification of UML, a lot of publications and work concentrated on this standard, paving the way for low-cost and open-source tools. ReqIF is hoped to encourage a similar development in the area of RE.

ReqIF allows structuring of requirements with many attributes and supports hierarchical structures and links between these requirements as well [8]. Eclipse RMF is an open-source implementation of the ReqIF standard [6]. In industry, there are a lot of customization of proprietary tools but RMF is built on the open ReqIF standard that is currently being adopted by commercial tools [8]. The proposed tool will rely on an emerging, solid standard by building on the RMF.

While the underlying data model is safely built on a standard, additional project specific features could be built on top. The proposed tool should incorporate such features to encourage involvement of the nonengineer stakeholders to the requirements elicitation. What is meant by additional features can simply be a user-friendly GUI, instead of ProR, that creates simple diagrams of requirements with their attributes and links, provides traceability of requirements, or makes suggestions between similarly titled requirements etc. This UI could then be used within workshops or individually, and with RE specialist supervision or without it. A medium of solid recording for requirements that supports nonengineer stakeholders’ capabilities could be beneficial for better requirements elicitation through more freedom in scheduling and greater direct contribution from stakeholders. Of the tacit knowledge class definitions, the third class of requirements would benefit from increased direct involvement by the stakeholders since they are the ones who withhold this class of information while the RE specialist is unaware of the situation. The fourth class of requirements is where both the RE specialist and other stakeholders are unaware and the tool, with its proposed visualization and traceability features, could help stimulate further investigation by both parties.
3.4.1.1 Eclipse Requirement Modeling Framework GUI: ProR

The Eclipse RMF has an Eclipse plugin called ProR [8]. This plugin constitutes a GUI inside Eclipse for creating and modifying ReqIF specifications. Since ProR is an Eclipse plugin, the possibility of usage by nonengineer stakeholders is limited. It is not designed with a nonengineering audience in mind. So, the solution proposed within this thesis is to develop a new tool that uses RMF data models for ReqIF but independent of ProR; has its own GUI and is online which opens it to many extra features such as remote access, collaboration, and diagram creation.

The collaboration mentioned here is an in-place collaboration such as a forum collaboration. Of course, when using ProR to create ReqIF specifications, these specification files are ultimately created to be shared between different organizations (e.g., automobile brand factories and automobile-part factories). However, with the designed tool collaboration as the file is being specified is more important rather than sharing a finished specification with another organization.

A relevant advantage of ProR over the proposed tool is that it works directly on the implemented ReqIF classes. This is advantageous because the implementation of a specialized GUI would need to have its own corresponding data model to bind with ReqIF data model. For example, the designed web-based tool will need to convert these data types or adapt new data types for web-specific use (e.g., JavaScript classes). This will produce much overhead for the tool if it is to implement correspondence with the whole standard. Although this is a definite drawback for the designed tool, it is arguable that the benefits brought in by the tool as such could overweigh the extra work associated with its data model communication.

3.4.2 Client-side: AJAX, JSON, and JavaScript

AJAX stands for Asynchronous JavaScript and XML. It is a collection of techniques used to create asynchronous web applications. Asynchronous web applications depend on partial updates of pages instead of full page updates with HTML data received from the server.

Before the popularization of asynchronous web applications, every HTTP request resulting from a user action loaded a complete HTML page from the server. This resulted in poor user experience as each action made the web page disappear and reappear on the client-side. The term AJAX represents a group of Web technologies that are used to implement a Web application that communicates with a server in the background, without interfering with the current state of the page. In the article that coined the term AJAX [30], it is explained that the following technologies are incorporated:

- HTML (or XHTML) and CSS for presentation
• The Document Object Model (DOM) for dynamic display of and interaction with data
• XML for the interchange of data, and XSLT for its manipulation
• The XMLHttpRequest object for asynchronous communication
• JavaScript to bring these technologies together

However, the technologies that are mentioned in this article have advanced since the publication of this article in 2005. Now XML and XSLT is not required. In this thesis implementation, JavaScript Object Notation (JSON) will be used for data exchange between server and client-sides. JSON is a lightweight, text-based, language independent data interchange format[31]. It is chosen to be used as the data interchange format in this thesis project.

jQuery is a popular general purpose JavaScript library that is used by other JavaScript libraries as well. By using jQuery, asynchronous HTTP requests will be made with JSON payload. The data received from the server-side will be kept in JavaScript classes with KnockoutJS bindings. KnockoutJS is a JavaScript library that focuses on tying HTML element with JavaScript models. It achieves a clean implementation of data model and UI communication on the client-side. It also has JSON encoding/decoding capabilities which will be used. Lastly, JointJS will be used to dynamically create diagrams on HTML5 canvas element. It is open-source and some custom diagram elements will be extended from some base classes in JointJS.

3.4.3 Server-side: Java Servlets & Jackson

A ”servlet” is a Java class that responds to requests through HTTP or other protocols. In this project, some Java servlets will be deployed on an Apache Tomcat server. They will listen client requests across HTTP. They will perform operations on the data model while providing information to the client-side.

In the previous section, it is described that JSON will be used as the data interchange format. On the client-side, KnockoutJS and native JavaScript parsers are used for handling JSON. On the server-side, this task is performed by the Java library, Jackson. It is an open-source JSON processor. The serialized JSON strings will be exchanged through the content payload of HTTP requests and responses between the server and client-side. For example, a new document that is created on the client-side will be serialized by JavaScript classes, sent with an AJAX request over to the server-side. This document will then be deserialized into a Java object and stored by the server with an assigned ID. Then, this newly assigned ID will be written on the response message of the server to the client.
### Chapter 3. Project Design

#### Figure 3.3: GUI grids mock-up

<table>
<thead>
<tr>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document A</strong></td>
</tr>
<tr>
<td><strong>Document B</strong></td>
</tr>
<tr>
<td><strong>Document C</strong></td>
</tr>
</tbody>
</table>

#### Field Requirements of **Document A**

<table>
<thead>
<tr>
<th>Field 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 2</td>
</tr>
<tr>
<td>Field 3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

#### Other Requirements of **Document A**

<table>
<thead>
<tr>
<th>Req. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. 2</td>
</tr>
<tr>
<td>Req. 3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Figure 3.4: Requirement links mock-up
Figure 3.5: Diagram mock-up
Chapter 4

Project Implementation

In this chapter, the implementation of the thesis tool is presented. First, the overall structure of the tool will be explained. Then each part will be explained in more detail. Finally, the difficulties encountered during the implementation will be presented.

4.1 Overall Architecture

The system is a web-based application. It comprises a server-side and a client-side UI which interacts with it. On the client-side, the user interacts with the data and on the server-side, a Java servlet catches the requests and sends back responses accordingly. Figure 4.1 shows the overall architecture in a diagram.

Data model of the architecture resides on top of the RMF implementation of ReqIF. A level of abstraction is built on top of ReqIF that is to be serialized into JSON messages. This data is communicated by Java servlets through AJAX calls. Data is communicated as JSON objects between the server and the client-side. The received JSON objects are reflected onto the client browser through KnockoutJS, the JavaScript library. From this data, diagrams are created by using another JavaScript library, joint.js.

4.2 Data Model

As mentioned before, ReqIF is a standard for recording requirements. ReqIF allows the structuring of natural language artifacts and supports an arbitrary number of attributes and the creation of attributed links between artifacts [8]. In this way it provides an organization framework to effectively record and easily track requirements. It is also mentioned in [8] that the requirements can be arbitrarily grouped into document-like
constructs which is what the tool is designed to do. Data model of the thesis tool is built on top of RMF implementation of ReqIF; that is, wrapper classes for some ReqIF classes are implemented. The wrapper classes are namely ReqIFRoot, DocumentReqIF, SpecObjectReqIF, and SpecRelationReqIF.

Figure 4.2 shows a UML class diagram of the whole data model architecture. The ReqIF implementation of RMF is denoted by the package RMF ReqIF. These classes are referenced by the four wrapper classes. The class DocumentRepository functions as the database for the web-application. This class is used by the server-side classes (Servlets) to store and retrieve data.

Before going into the wrapper classes, the ReqIF classes that are used and/or referenced will be explained. ReqIF constitutes an XML structure. At the top of the XML tree, is the element called ReqIF. It is simple a placeholder for the whole requirements document. The important field of this class is ReqIFContent which holds all the following requirements data inside itself. In the ReqIF specification, a SpecObject represents a requirement. A SpecObject can have a SpecType which is used as a type indicator. A Specification holds SpecHierarchy elements which refers to one SpecObject each. In this way, SpecObjects are grouped and organized. Using referencing of SpecHierarchy elements, the same SpecObject can be referenced from various SpecHierarchies and also from various Specifications. A SpecRelation represents a relation between two SpecObjects; a source and a target.
Specification, SpecObject, and SpecRelation can have associated type objects in ReqIF; SpecificationType, SpecObjectType, and SpecRelationType respectively. These type objects of ReqIF are used to create a template of fields for their respective objects. For example, a SpecObject has a number of AttributeValues, which hold the actual content of the SpecObject. A SpecObjectType also has a list of Attributes which gives these Attributes to a SpecObject when it is set as the type of that SpecObject. For simplicity, and because of lack of need for the design at hand, these Attributes of SpecObjects are not used. Instead, only the definition fields are used as requirements descriptions. Similarly, type objects are not used as templates for Attributes but rather as a simple type indicator string.

These mentioned ReqIF classes are used in specific parts of the data model necessary to collect requirements for the selected project domain, ECM document management. The documents are organized separately with their field and other requirements. The corresponding ReqIF class for these documents are Specifications. This allows for a document specification to reference requirements which are backed by SpecObjects themselves. The relations between requirements are represented by SpecRelation objects. To summarize;

1. ReqIF element sits on top of the whole requirements specification.
2. Specification groups the SpecObject requirements into documents that correspond to physical documents that the ECM system is designed to contain.
3. SpecObject represents an actual requirement (field or other). Its description field is used for storing natural language requirements data.
4. A SpecRelation is used to link together two requirements. It also has its own description.

In addition to these, SpecHierarchy objects are used with the Specifications but since they are not needed to be communicated outside the server-side, they do not have a corresponding wrapper class. There are two SpecType objects that are used to denote if a SpecObject is a field requirement or an other requirement. Also, a single SpecificationType object is used for all documents. It denotes that the Specifications are 'document’s. For the same reason as the SpecHierarchy class, these classes do not have wrapper classes either.

The corresponding wrapper classes for the enumerated ReqIF classes are:

1. RootReqIF
2. DocumentReqIF
3. SpecObjectReqIF
4. *SpecRelationReqIF*

These wrapper classes are implemented to provide a level of abstraction that hides the details of ReqIF implementation while making JSON serialization possible. This will be discussed in detail later on.

DocumentRepository serves as the database of the application. It is obviously not meant to be included in a production version of the tool as it provides static data that is initialized on each server reset. It is meant to be useful only in the proof-of-concept that is presented with this thesis. Functions `getNewDefaultDocument` and `getSomeSpecRelations`, together with a piece of static code, populate initial data into the ReqIF data model that is represented by attribute `root` of type `RootReqIF`.

According to ReqIF specification, ReqIF objects should be provided with unique IDs. Next three functions `getUniqueDocID`, `getUniqueSpecID`, and `getUniqueRelationID` provide unique IDs for ReqIF objects. The ID creation does not necessarily need to be implemented inside the database; these function are inside DocumentRepository class only because it is convenient and their placement is not crucial. Next three groups of functions correspond to creating, updating, and deleting documents, requirements, and relations. Finally, it provides a getter function for the ReqIFRoot object. These functions are called from the servlets with the exception of `getSpecObject` function which is called from the SpecRelationReqIF class to find SpecObjects by their IDs.

*RootReqIF* class represents the whole of a requirements specification. It holds the top-level ReqIF element together with a list of DocumentReqIF and SpecRelationReqIF objects. `getHeaderTitle` and `setHeaderTitle` functions operate on ReqIFHeader attribute of the ReqIF element which is expected to contain the name of the specification (e.g. general project name). There are functions that work with the corresponding functions in DocumentRepository to create, update, and delete elements. There are getter functions for documents and relations.

*DocumentReqIF* class encapsulates requirements for a document that is to be included in the ECM system. It defines a static document type (SpecificationType) that is currently the same for all documents since there is no distinction implemented between documents. Another two static types (SpecObjectType) are defined for requirements that the document holds. These are set as types of individual requirements. The document object is of Specification class and it is held by the field `document`. There are two lists, `fieldRequirements` and `otherRequirements`, that hold the field and other requirements respectively. There are functions for inserting, updating and removing requirements and finally getter functions for requirement lists and the document itself.

*SpecObjectReqIF* class wraps a single requirement. The SpecObjects are stored under documents and sometimes it is needed to reach their DocumentReqIF parents. So, IDs for their parent documents are held in SpecObjectReqIF in the attribute `docId`. The type of SpecObject is set to one of the two static SpecObjectTypes defined in DocumentReqIF, namely, `fieldType` and `otherType`. 

SpecRelationReqIF class wraps a SpecRelation which defines a relation between two requirements. The source and target SpecObjectReqIF objects are held by this class. However, getter/setter functions for the source and target return/take string IDs. This is done to relive JSON from serializing these SpecObjectReqIF objects twice since they are already being communicated once with the client through DocumentReqIF objects.

It should be noted here that the wrapper classes hold relevant objects under hierarchies both in their ReqIF references and in their own lists. For example, in ReqIFRoot class, DocumentReqIF objects that are inserted are kept in the list docs of RootReqIF while the Specification objects that these DocumentReqIF objects hold are added to the root ReqIF object as well. The same goes for the requirements kept in DocumentReqIF objects; they are stored in both the Specification object (as SpecObjects) and in separate lists inside DocumentReqIF (as SpecObjectReqIF). This is done to allow JSON to perform more efficiently by reducing the size of the serialized JSON string output. This will be mentioned in more detail in the next section.

### 4.3 Server-side Implementation

The server-side implementation comprises the Java servlets that respond to client requests, response objects, and a JSON encoder/decoder. These classes can be seen in Figure 4.3.

There are four servlet classes implemented that extend HttpServlet class of Java: ReqIFServlet, DocumentServlet, SpecObjectServlet, and SpecRelationServlet. These servlets share the workload associated with calls regarding each of the four wrapper classes described in the previous section. They work with the backend data model which is represented by the package Backend in Figure 4.3.

RequestContext class wraps request handling functionality in itself. A RequestContext is created for each call that reaches a servlet. It takes a HttpServletRequest and a HttpServletResponse objects as references to its constructor and saves references to these objects in its attributes request and response respectively. References for the InputStream of request and the OutputStream of response is created. A Map is used for keeping parameters passed with a request. The function getRequestContentAs is used to read objects from the AJAX content of the requests while the function writeToResponseJson writes response objects to the response stream. These two functions work together with the class JsonEncoder so they are generically typed as well.

JsonEncoder class uses the JSON processor library included in the project, Jackson [32]. It is a simple generic class that encodes/decodes objects into/from JSON strings. In its constructor, it takes the type of objects that are going to be used with it. It uses the ObjectMapper class from Jackson library to perform these two operations on the four wrapper classes previously described. When the ObjectMapper is provided with an object to serialize (i.e. turn into a JSON string), it gathers the attributes through
the getter functions of the object being serialized. The same goes for deserialization (i.e. create Java objects from JSON string) through the use of setter functions. The getter/setter functions in wrapper classes are set such that the size of JSON strings are minimalized.

*Response* class defines an object for communicating response information back to the client after a request. It has a message and a boolean to denote whether the request has been successfully carried or not. There are two special classes that extend Response class: *GetRootResponse* and *SaveResponse*. *GetRootResponse* is used when the client first opens the web page. It is used to send the RootReqIF object over to the client-side. After this initiation, no complete object data is sent to the client. However, there are two special cases when creating and updating new objects. When creating a new object, all the information is provided by the client-side except the ID. The ID is uniquely assigned by the DocumentRespository. This ID is needed to be sent back to the client so it can be stored and displayed on the client-side. *SaveResponse* class is used for this purpose. The ID of the newly created object is stored in a *SaveResponse* object and it is sent after serialized by *JsonEncoder* class. The *saveType* is also set to *INSERT*. The second case where *SaveResponse* is needed is when an update is being requested. In this case the attribute *saveType* is used to mark whether any information has been updated or not (i.e. *UPDATED* or *NONE*).

### 4.4 Client-side Implementation

Client-side implementation consists of a single web page governed by some JavaScript classes that use third-party JavaScript libraries. The used libraries are *KnockoutJS* [33], *JointJS* [34], and *jQuery* [35]. Figure 4.4 shows a UML class diagram of the client-side class structure.

As mentioned in the previous chapter, KnockoutJS, allows for easy binding of JavaScript objects with HTML Document Object Model (DOM) objects. It has a Model-View-View Model (MVVM) architecture. MVVM is similar to MVC but is designed specifically for UIs. Model and View in MVVM corresponds to the domain model and GUI as in the classic MVC architecture. The difference of MVVM lies on the idea of View Model. This layer of the architecture is a mediator between the domain data and their representation on the View. It provides data binding between View Model and View. In a way, it is a specialized Controller.

In Listing 4.1, an example usage of KnockoutJS is presented with a piece of simplified code from the implementation. The function Document implements a simple JavaScript class that holds two attributes; id and description. It can also be instantiated with existing document info. ViewModel is the View Model implementation which holds the objects to be bound to the View and relevant functions. In this case, it holds a single Document objects and defines a function that makes an AJAX request to a Java servlet. On the last three lines of the script element, a ViewModel object is created,
supplied to KnockoutJS to apply bindings for, and an AJAX call is made with jQuery to request the document from the server-side. KnockoutJS introduces an HTML attribute `data-bind`. Using this attribute, View Model objects are bound with the HTML elements of the View. The `<div>` element is bound with `document` while the text of a span element and the text of an input element is bound with `id` and `description` of document respectively. From this point on, if the description is modified through the input, the attribute of the Document object will be automatically updated. The opposite also holds true; if the description attribute is changed, for example, by activities of another function, the input element on the HTML will be updated as well. This is achieved through KnockoutJS’s observable JavaScript objects that detect changes and notify their observers. The line that creates the `document` object is defined as `ko.observable(new Document())` initiates it as a KnockoutJS observable.

```javascript
function Document(document) {
  this.id = '';
  this.description = '';
  if (document) {
    this.id = document.id;
    this.description = document.description;
  }
}

var ViewModel = function() {
  var self = this;

  self.document = ko.observable(new Document()); // An observable variable

  self.getDocument = function() {
    $.ajax({
      url: 'Servlet',
      type: 'GET',
      success: function(responseData, textStatusStr, jqXHR) {
        if (responseData.successful) {
          self.document(responseData.document);
        } else {
          alert('Unsuccessful: ' + responseData.message);
        }
      },
      error: function(jqXHR, textStatusStr, errorStr) {
        alert(errorStr + ' error at getDocument');
      }
    });
  }

  var viewModel = new ViewModel();

  ko.applyBindings(viewModel); // This starts Knockout bindings
  viewModel.getDocument();
}
```
Listing 4.1 presents a simple example of how KnockoutJS is used together AJAX to construct the client-side structure. A UML diagram for the actual client-side implementation of the tool is presented in Figure 4.4 and it basically consists of a complicated version of the presented example in action.

Figure 4.5 shows a screenshot of the implemented tool. It consists of a table for defined documents. When a row of this grid is selected, the field and other requirements associated with that particular document is listed on two separate grids underneath the first one. When a row of one of the requirement grids is selected, a table that lists the relations concerning that requirement is listed on a grid that appears below that row. Under each grid, there is a button for adding a new row (i.e. creating a new object) and a save button that will create an AJAX call to the server to save the current state of the grid. Also, on each row there is a delete button while the fields of objects are modifiable in place.

*RootModel* is the View Model that holds objects that are displayed and modified through the HTML page. It holds instances of *Root, Document, SpecObject,* and *SpecRelation* classes while the HTML elements is bound to this data. *RootModel* holds rootObj object, documents array, and relations array as KnockoutJS observables. It also holds observable objects selectedDocument and selectedSpecObject to keep track of selected rows of respective grids. showedRelations object is another observable array that is used to calculate and store relations related to a SpecObject when its corresponding row is selected on its grid.

setRoot function sets the values of rootObj, documents, and relations after receiving a response for the AJAX call made through the function getRoot. Functions starting with "add" are called when the "Add ..." buttons below grids are clicked. They add a new object into their respective arrays; one of documents, relations, or one of requirement arrays in Document class. The two functions starting with "select" are called when a document or requirement row is selected on the grids. They set the selectedDocument and selectedSpecObject so they can be bound to appearing requirements and relations grids. getDocumentByID is a utility function
used when showedRelations is populated by selectSpecObject. The purpose of dummy is explained on Chapter 5. The rest of the functions, except the ones starting with "ajax" and generateDiagrams, are called when a save or delete button click occurs.

Functions starting with "ajax" are called by these functions to communicate with the server-side by sending HTTP requests with JSON content. For example, in the function ajaxSaveDocument, the Document parameter is passed when the HTML button’s click event occurs. The passed Document object is serialized into JSON and sent over to the server-side. When the server-side returns the ID assigned to the newly inserted Document object, this ID is assigned to the passed parameter on the success callback method of the AJAX call (Listing 4.2). Also on the success callback, alerts are shown according to the received response type. As mentioned in the previous section, server marks the type of the result as NONE, INSERTED, and UPDATED accordingly. On success callback functions of delete functions’ AJAX calls, the actual removal of the objects from the View Model is done. This is done to avoid prematurely deleting these objects before the deletion is actually performed on the server so the data is not lost in case of a miscarriage during the deletion on the server-side.

```javascript
self.ajaxSaveDocument = function(document) {
  $.ajax(
    { url : 'DocumentServlet',
      type : 'POST',
      contentType : 'application/json; charset=utf-8',
      dataType : 'json',
      data : JSON.stringify({
        id : document.id(),
        description : document.description,
        fieldRequirements : document.fieldRequirements(),
        otherRequirements : document.otherRequirements()
      }),
      success : function(responseData, textStatusStr, jqXHR) {
        if (responseData.successful) {
          document.id(responseData.id);
          if (responseData.saveType == 'INSERTED')
            alert("Inserted " + responseData.id);
          else if (responseData.saveType == 'UPDATED')
            alert("Updated " + responseData.id);
        } else {
          alert("Save unsuccessful! ' + responseData.message);
        },
        error : function(jqXHR, textStatusStr, errorStr) {
          alert(errorStr + ' error at ajaxSaveDocuments');
        }
      });
};
``` 

LISTING 4.2: An example AJAX caller function
The four classes, `Root`, `Document`, `SpeObject`, and `SpecRelation` classes hold the crucial information. Document class’s attribute `id` is an observable object. When a new `Document` object is being created, it does not have an ID. Documents are assigned IDs when they are saved by the server. The assigned IDs are sent back from the server on responses and written on `id` attributes. Since `id` attribute is an observable, the View is updated to show IDs when they are received and written on `ids`. The observable arrays `fieldRequirements` and `otherRequirements` hold requirements as `SpecObject` objects while `allRequirements` hold the composition of these two arrays. `allRequirements` is a special kind of observable, a `computed observable`, which means it is computed from other observable objects. `SpecObject` and `SpecRelation` holds requirements and relations respectively.

All these four classes, they are parallel to their counterparts in the server-side, that is, the four wrapper classes. However, they have some extra, client-side specific attributes. `allRequirements` attribute of the `Document` class was one of them. `SpecRelation`’s `linkedDoc`, `linkedFrom`, and `linkedTo` are others. A relation links two requirements together; source and target. So when listing relations on the grid of the View, they are listed under both these requirements. According to the selected requirement row, source or target, the "Linked Document" and "Linked Requirement" fields show the information of the other requirement. `linkedDoc` and `linkedTo` keeps this other requirement’s information while `linkedFrom` hold the selected requirement.

The last function on the `RootModel` class is `generateDiagram`. This function is called on the "Diagram" button’s click event. `generateDiagram` creates JointJS objects from the information present in the View Model. JointJS is a diagramming library that uses the canvas element of HTML5. It has defined diagram element classes that are base classes like rectangles or links or special purpose classes to be used for creating specific types of diagrams such as ER diagrams, UML class diagrams, logic circuits, etc. In this implementation, three classes are extended from base classes of JointJS: `reqif.Document` and `reqif.Requirement`. The purposes of these classes should be obvious. JointJS’s default class for links, `joint.dia.Link`, is used to display relations. IDs of relations are put on the links but since the descriptions would obstruct the diagram greatly, `reqif.RelationLegend` class is designed to create a legend of descriptions. An example diagram is shown in Figure 4.6. The documents are draggable. Also, the shape of the links are adjustable so they can be reshaped to look better.

### 4.5 Technical Issues Encountered

#### 4.5.1 Data Model: ReqIF & View

As mentioned before, ProR implementation resides directly on top of the ReqIF implementation of RMF and it is advantageous because of the lack of need for intermediate class structures: the wrapper classes of the server and the JavaScript classes that hold
the ReqIF object information. Some confusion occurred when designing and implementing these classes. ReqIF standard objects have numerous attributes implementing capabilities that are not implemented in this thesis project. The purpose of wrapper classes was hiding unused attributes of ReqIF standard so they are not serialized into JSON. However, as later discovered, maybe JSON serialization of the ReqIF objects could have been customized to exclude unwanted attributes instead of creating wrapper classes. On the other hand, there would still be a need for an intermediary class where some validations could be implemented. Another observed benefit of the wrapper classes was their parallel structure with the JavaScript classes of the View Model. JavaScript classes are already needed to implemented a good UI structure. When the ease of converting JSON-serialized wrapper classes to these classes, the tradeoff of the wrapper classes became more rationalized.

4.5.2 JSON

At first attempts of the implementation, KnockoutJS’s built-in JSON serializer was used to serialize objects which contain observable attributes (Listing 4.3). This serializer automatically produced a serialization based on all the attributes of a given object. If there were some observable objects present, they were also ’unpacked’. An observable object is a KnockoutJS object that is wrapped around the regular JavaScript object it is stored with. To reach the actual object held by an observable object, it needs to be unpacked by () . This returns the actual content of an object. The JSON serializer of KnockoutJS did these unpacking operations (there are observables nested inside one another also) automatically as well. However, this raised exceptions from the JSON processor on the server-side. The problem was that some intermediary attributes that are used exclusively for View purposes, such as allRequirements observable array of Document class, had no matching property in the Java wrapper classes residing on the server-side and so the JSON processor raised exceptions.

```javascript
self.ajaxSaveDocument = function(document) {
  $.ajax({
    ...
    data : ko.toJSON(document),
    ...
  });
};
```

LISTING 4.3: Old JSON serialization

A property on the static ObjectMapper instance of JsonEncoder class is set to bypass enforcing every property to be deserialized. If some fields come with the request JSON data that cannot be matched with the classes it is deserialized to, they raise an exception during the deserialization of data into Java objects. This kind of data was preferred to be discarded and so this exception is circumvented by setting a flag on ObjectMapper instance to ignore such cases. After consideration, it was found out that if the JSON serializer of jQuery is used, this flag is not needed to be set. Using jQuery, data field
of the AJAX call is set with attributes of the Document object omitting the problematic allRequirements array (Listing 4.4). The same problem was present for the linkedDoc, linkedFrom, and linkedTo attributes of the SpecRelation class as well.

```javascript
self.ajaxSaveDocument = function(document) {
  $.ajax({
    ...
    data : JSON.stringify({
      id : document.id(),
      description : document.description,
      fieldRequirements : document.fieldRequirements(),
      otherRequirements : document.otherRequirements()
    }),
    ...
  });
};
```

**LISTING 4.4: Fixed JSON serialization**

### 4.5.3 Customizing JointJS

JointJS has many different ready-to-use sublibraries for specific purposes. However, creating custom classes proved to be not so easy. There is an API entry on JointJS website [34] but it does not really constitute an API documentation but rather a guideline. Lack of adequate API and also lack of commented code, it became very hard to modify default behavior of classes. For example, join.dia.Link class comes with functionality, through the canvas, such as deleting the link or changing its source and target connections to different elements. These features were to be removed from this implementation as the other parts do not support it. Since the code is not easily reusable, the way to do necessary modifications was to modify CSS. The difficulties of JointJS suggests that it is worth considering another diagramming library in future.
Figure 4.2: UML class diagram for the data model
Figure 4.3: UML class diagram for the server-side implementation
Figure 4.4: UML class diagram for the client-side implementation
**Figure 4.5:** A screenshot of the finished implementation
Figure 4.6: A screenshot of a generated diagram
Chapter 5

Tests

5.1 Test Design

The UI of the implementation was tested. First, a risk analysis is done to list possible risks with the UI. Then, test suites to test for these lists are established. Lastly, specific test cases and scenarios are designed.

5.1.1 Identifying Risks

After consideration, four classes risk are defined: functional risks, usability risks, efficiency risks, and portability risks. Functional risks include suitability and accuracy. Suitability covers the functionality of the interface on the level of functioning and displaying correctly. Accuracy risks include accepting incorrect data or creating incorrect output (i.e. diagram, grids). The next class of risks, usability, lists understandability, learnability, and attractiveness as sub-risks. Understandability risk is concerned with the users’ inability to understand the purpose of the interface. Learnability is concerned with inability to learn how to use the interface and attractiveness risk is that the interface is not visually appealing to the users. The third class of risk class is efficiency with a single sub-risk; time behavior. It is the risk that the response time of the interface might be too long and obstruct the flow of usage. Last class of risks, portability, is concerned with the adaptability of the interface across different platforms (i.e. browsers). These risk classes are listed in Table 5.1.

In Table 5.1, at the columns "Technology Risk" and "Business Risk", the risk potential values are assigned between 1 (highest) and 5 (lowest). "Risk Priority" values are the multiplications of these two columns to reach a final indicator of priority. It’s range values are listed in Table 5.2. These values are used in deciding what test suites will be designed and also the extent of actual testing that will be done to cover defined risks.
### Table 5.1: Identified Risk Classes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Functional System Risks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1 Suitability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface does not work.</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Extensive</td>
</tr>
<tr>
<td></td>
<td>Interface not well displayed</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>Opportunity</td>
</tr>
<tr>
<td></td>
<td>(resolution problems or font</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sizes, etc.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.2 Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data is not validated.</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>Extensive</td>
</tr>
<tr>
<td></td>
<td>Diagram is wrong.</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>Extensive</td>
</tr>
<tr>
<td>2</td>
<td>Usability System Risks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.1 Feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Users do not receive appropri-</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>Extensive</td>
</tr>
<tr>
<td></td>
<td>ate feedback.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.2 Understandability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Users are not able to under-</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>Broad</td>
</tr>
<tr>
<td></td>
<td>stand the aim of the interface.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.3 Learnability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Users are not able to un-</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>Broad</td>
</tr>
<tr>
<td></td>
<td>derstand how the interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>works.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4 Attractiveness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Users do not like the interface</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>Cursory</td>
</tr>
<tr>
<td>3</td>
<td>Efficiency System Risks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.1 Time behavior</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response time is too long (UI</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>Opportunity</td>
</tr>
<tr>
<td></td>
<td>interaction or diagram genera-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tion)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Portability System Risks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.1 Adaptability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface could be affected by</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>Opportunity</td>
</tr>
<tr>
<td></td>
<td>using different browsers.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.1.2 Test Suites

Table 5.3 lists the test suites that are decided to be created. There are two test suites that address risks 1 and 2 (Table 5.1). Usability risks, except feedback, are omitted because of the lack of real users for the tool. Feedback risks are tested alongside functional risks. A test suite that addresses the all usability risks (e.g. with the SUMI and WAMMI standard surveys) should be designed after acquiring a user base. Similarly, because a production implementation is not present, efficiency risks are not addressed neither. The time behavior is most prevalent when a remote server is being used. The other
### Extensive 6 - 10

### Broad 11 - 15

### Cursory 16 - 20

### Opportunity 21 - 25

### Report Bugs

**Table 5.2: Risk Priority Ranges**

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Test Objectives</th>
<th>Risk Coverage</th>
<th>Sequence Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Functionality</td>
<td>Verify all the functionalities of the interface. Find bugs in the interface logic. Ensure that there are no problems during the execution in the browser.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Portability</td>
<td>Verify the adaptability of the interface</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 5.3: Designed Test Suites**

Operations are done on the client-side with JavaScript so they are instantaneous. Also, the server communication is done with asynchronous calls so no obstructions due to latency is expected.

#### 5.1.3 Test Cases and Scenarios

As explained in Chapter 4, the interface consists of four grids/forms:

- Document Form
- Field Requirement Form
- Other Requirement Form
- Relation Form

Tables 5.4, 5.5, 5.6 and 5.7 show test cases to apply on individual forms to test them. The listed tests are similar across all four forms. There is a click-row operation to be tested on all forms (except Relation Form) that should select a row and list its children objects. The operations addition, updating, and deletion are tested on all forms. There is also a pre-condition of number of grid-rows. Without any rows present, some outputs are changed and, since “Delete” buttons are located on each row, deletion is disabled.
<table>
<thead>
<tr>
<th>Case No.</th>
<th>Input</th>
<th>Pre-condition # of requirements</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on a document row</td>
<td>&gt;0</td>
<td>The related requirements are displayed</td>
</tr>
<tr>
<td>2</td>
<td>Button ”Add Document“</td>
<td>&gt;0</td>
<td>Add a new empty document row with ID assigned</td>
</tr>
<tr>
<td>3</td>
<td>Button ”Save“</td>
<td>&gt;0</td>
<td>Updates are performed (if there are any) and feedback is given</td>
</tr>
<tr>
<td>4</td>
<td>Button ”Diagram“</td>
<td>&gt;0</td>
<td>The diagram is displayed. Feedback is present.</td>
</tr>
<tr>
<td>5</td>
<td>Button ”Delete“</td>
<td>&gt;0</td>
<td>The row is deleted and feedback is given.</td>
</tr>
<tr>
<td>6</td>
<td>Button ”Add Document“</td>
<td>0</td>
<td>Add a new empty document row with ID assigned</td>
</tr>
<tr>
<td>7</td>
<td>Button ”Save“</td>
<td>0</td>
<td>Nothing happens</td>
</tr>
<tr>
<td>8</td>
<td>Button ”Diagram“</td>
<td>0</td>
<td>Nothing happens</td>
</tr>
</tbody>
</table>

**Table 5.4: Test Cases for Document Form**

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Input</th>
<th>Pre-condition # of requirements</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on a field requirement row</td>
<td>&gt;0</td>
<td>The related relations are displayed</td>
</tr>
<tr>
<td>2</td>
<td>Button ”Add Field Requirement“</td>
<td>&gt;0</td>
<td>Add a new empty field requirement row with ID assigned</td>
</tr>
<tr>
<td>3</td>
<td>Button ”Save“</td>
<td>&gt;0</td>
<td>Updates are performed (if there are any) and feedback is given.</td>
</tr>
<tr>
<td>4</td>
<td>Button ”Delete“</td>
<td>&gt;0</td>
<td>The row is deleted and feedback is given.</td>
</tr>
<tr>
<td>5</td>
<td>Button ”Add Field Requirement“</td>
<td>0</td>
<td>Add a new empty field requirement row with ID assigned</td>
</tr>
<tr>
<td>6</td>
<td>Button ”Save“</td>
<td>0</td>
<td>Nothing happens</td>
</tr>
</tbody>
</table>

**Table 5.5: Test Cases for Field Requirement Form**
Chapter 5. Results

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Input</th>
<th>Pre-condition # of relations</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on a other requirement row</td>
<td>&gt;0</td>
<td>The related relations are displayed</td>
</tr>
<tr>
<td>2</td>
<td>Button ”Add Other Requirement”</td>
<td>&gt;0</td>
<td>Add a new empty other requirement row with ID assigned</td>
</tr>
<tr>
<td>3</td>
<td>Button ”Save”</td>
<td>&gt;0</td>
<td>Updates are performed (if there are any) and feedback is given.</td>
</tr>
<tr>
<td>4</td>
<td>Button ”Delete”</td>
<td>&gt;0</td>
<td>The row is deleted and feedback is given.</td>
</tr>
<tr>
<td>5</td>
<td>Button ”Add Other Requirement”</td>
<td>0</td>
<td>Add a new empty field requirement row with ID assigned</td>
</tr>
<tr>
<td>6</td>
<td>Button ”Save”</td>
<td>0</td>
<td>Nothing happens</td>
</tr>
</tbody>
</table>

Table 5.6: Test Cases for Other Requirement Form

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Input</th>
<th>Pre-condition # of relations</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on a relation row</td>
<td>&gt;0</td>
<td>Nothing happens</td>
</tr>
<tr>
<td>2</td>
<td>Button ”Add Relation”</td>
<td>&gt;0</td>
<td>Add a new empty relation row with ID assigned</td>
</tr>
<tr>
<td>3</td>
<td>Button ”Save”</td>
<td>&gt;0</td>
<td>Updates are performed (if there are any) and feedback is given.</td>
</tr>
<tr>
<td>4</td>
<td>Button ”Delete”</td>
<td>&gt;0</td>
<td>The row is deleted and feedback is given.</td>
</tr>
<tr>
<td>5</td>
<td>Button ”Add Relation”</td>
<td>0</td>
<td>Add a new empty relation row with ID assigned</td>
</tr>
<tr>
<td>6</td>
<td>Button ”Save”</td>
<td>0</td>
<td>Nothing happens</td>
</tr>
</tbody>
</table>

Table 5.7: Test Cases for Relation Form

The test scenarios listed in Figure 5.1 and Figure 5.2 represent two common scenarios that could take place. They test whether the four forms work together or not by covering a whole scenario rather than the individual test cases presented in Tables 5.4, 5.5, 5.6, and 5.7. First scenario designs an operation flow consisting of creating documents, child requirements, and a relation between these child requirements. All kinds of interactions with the UI, except deletion, can be simplified to the steps taken in this scenario. The deletion operations are covered with Scenario 2, with attention to preserving consistency while deleting elements. A deletion should not be permitted if an element has child objects (e.g. a document has requirements) or related objects (i.e. requirements linked together).
1. Create two documents, 1 and 2
2. Select document 1
3. Add a requirement, A
4. Select document 2
5. Add a requirement, B
6. Select requirement B and add a relation to requirement A
7. Create the diagram

*Expected outcome:* During step 6, document 1 and requirement A is present on the relation row. Saving is successful. The created diagram shows the two documents and their requirements with a link between them.

**Figure 5.1:** Scenario 1 - Creation of two documents, two requirements, and a relation

---

1. Repeat steps 1-6 of Scenario 1 (Figure 5.1)
2. Delete one of the requirements; (A or B)
3. Delete one of the documents; (1 or 2)
4. Delete the relation
5. Delete requirement A
6. Delete document 1

*Expected outcome:* For steps 2 and 3, to preserve consistency, the deletion should not be allowed. Feedback should be given to explain the error. After removing the relation on step 4, remaining steps should be allowed.

**Figure 5.2:** Scenario 2 - Preserving consistency during deletion operations

---

### 5.2 Results

The designed test cases and scenarios are performed on the tool. After collecting the initial results, solutions are proposed for the failed tests. After implementing the solutions, failed tests are performed a second time. Corresponding to the test tables, the two sets of results and solutions are presented in separate tables; Table 5.8, 5.10, 5.11, 5.9.

As can be seen from the first set of results for individual tests, majority of the problems were in displaying correct feedback after operations. A few other problems were present such as the double click events, order of showing the feedback and completing deletions. When a button or a description textbox was clicked, rows were selected/unselected as well which created much confusion. *clickBubble* binding of KnockoutJS is set to
### Table 5.8: Test Results for Document Form (Table 5.4)

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Expected Output</th>
<th>First Result</th>
<th>Proposed Solution</th>
<th>Second Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The related requirements are displayed</td>
<td>Success. Field and other requirements are displayed with matching titles on top</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Add a new empty document row with ID assigned</td>
<td>Success. Row added, with ID.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Updates are performed (if there are any) and feedback is given</td>
<td>Failed. Feedback is not informative enough.</td>
<td>Fix feedback message</td>
<td>Success</td>
</tr>
<tr>
<td>4</td>
<td>The diagram is displayed. Feedback is present</td>
<td>Failed. Diagram shown but feedback is missing</td>
<td>Add appropriate feedback</td>
<td>Success</td>
</tr>
<tr>
<td>5</td>
<td>The associated row is deleted and feedback is given</td>
<td>Failed. Feedback is shown before deletion is seen. Feedback is not informative</td>
<td>Move feedback alert after actual deletion and modify feedback content.</td>
<td>Success</td>
</tr>
<tr>
<td>6</td>
<td>Add a new empty document row with ID assigned</td>
<td>Success.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Nothing happens</td>
<td>Success. Save button is not enabled.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Nothing happens</td>
<td>Success.</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5.9: Test Results for Relation Form (Table 5.7)

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Expected Output</th>
<th>First Result</th>
<th>Proposed Solution</th>
<th>Second Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nothing happens</td>
<td>Success. A row is clicked and nothing happened</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Add a new empty relation row with ID assigned</td>
<td>Failed. No ID is set.</td>
<td>Fix setting ID.</td>
<td>Success</td>
</tr>
<tr>
<td>3</td>
<td>Updates are performed (if there are any) and feedback is given</td>
<td>Failed. Feedback is not informative enough.</td>
<td>Fix feedback message</td>
<td>Success</td>
</tr>
<tr>
<td>4</td>
<td>The associated row is deleted and feedback is given</td>
<td>Failed. Feedback comes before deletion and is not informative</td>
<td>Move feedback alert after actual deletion. Modify feedback content.</td>
<td>Success</td>
</tr>
<tr>
<td>5</td>
<td>Add a new empty relation row with ID assigned</td>
<td>Failed. No ID is set.</td>
<td>Fix setting ID.</td>
<td>Success</td>
</tr>
<tr>
<td>6</td>
<td>Nothing happens</td>
<td>Success. Save button is not enabled.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Case No.</td>
<td>Expected Output</td>
<td>First Result</td>
<td>Proposed Solution</td>
<td>Second Result</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
<td>The related relations are displayed</td>
<td><em>Success.</em> The relations are shown in the relation form that appeared below the selected row.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Add a new empty field requirement row with ID assigned</td>
<td><em>Success.</em></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Updates are performed (if there are any) and feedback is given.</td>
<td><em>Failed.</em> Feedback is not informative enough.</td>
<td>Fix feedback message</td>
<td><em>Success.</em></td>
</tr>
<tr>
<td>4</td>
<td>The associated row is deleted and feedback is given.</td>
<td><em>Failed.</em> Feedback comes before deletion and is not informative</td>
<td>Move feedback alert after actual deletion. Modify feedback content.</td>
<td><em>Success.</em></td>
</tr>
<tr>
<td>5</td>
<td>Add a new empty field requirement row with ID assigned</td>
<td><em>Success.</em></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Nothing happens</td>
<td><em>Success.</em> Save button is not enabled.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 5.10: Test Results for Field Requirement Form (Table 5.5)**

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Expected Output</th>
<th>First Result</th>
<th>Proposed Solution</th>
<th>Second Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The related relations are displayed</td>
<td><em>Success.</em> The relations are shown in the relation form that appeared below the selected row.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Add a new empty field requirement row with ID assigned</td>
<td><em>Success.</em></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Updates are performed (if there are any) and feedback is given.</td>
<td><em>Failed.</em> Feedback is not informative enough.</td>
<td>Fix feedback message</td>
<td><em>Success.</em></td>
</tr>
<tr>
<td>4</td>
<td>The associated row is deleted and feedback is given.</td>
<td><em>Failed.</em> Feedback comes before deletion and is not informative</td>
<td>Move feedback alert after actual deletion. Modify feedback content.</td>
<td><em>Success.</em></td>
</tr>
<tr>
<td>5</td>
<td>Add a new empty field requirement row with ID assigned</td>
<td><em>Success.</em></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Nothing happens</td>
<td><em>Success.</em> Save button is not enabled.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 5.11: Test Results for Other Requirement Form (Table 5.6)**
false to suppress the click events for the rows when a button or textbox is clicked. The empty function `dummy` of `RootModel` class is used to have a click binding so the click is registered and `clickBubble` takes effect. Also, there were major problems with correctly creating and updating relations. After proposed solutions were implemented, the failing tests were repeated and, as seen from "Second Result" columns, seen to be fixed.

In Table 5.12 and Table 5.13, results for scenarios are presented. The first results of individual tests (on forms) were shared by them as well. These failing results were omitted to prevent repetition. Instead, only new output was listed for the scenario results. Figure 5.12 show that all tests were successful for Scenario 1. However, there were major problems with Scenario 2. As seen in Figure 5.13, no deletion was prevented when there were child or related elements. Although this did not create immediate problems, when some of the individual tests were repeated, it was seen that these deletion operations put the system in an invalid state. The repeated individual tests, such as creating and updating elements, suffered from these allowed deletions and hence rendered scenario tests unsuccessful. After the proposed solutions are applied, second round of testing for Scenario 2 has created successful results.
<table>
<thead>
<tr>
<th>Step No.</th>
<th>Step Content</th>
<th>First Result</th>
<th>Proposed Solution</th>
<th>Second Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create two documents, 1 and 2</td>
<td><em>Success</em>. Documents are created. Descriptions are updated and saved.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Select document 1</td>
<td><em>Success</em>. Document is selected and requirement forms are displayed.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Add a field requirement, A</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Select document 2</td>
<td><em>Success</em>. Document is selected and requirement forms are displayed.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Add another requirement, B</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Select requirement B and add a relation to requirement A</td>
<td><em>Success</em>. A new row is created. ID is set. Document 1 and its requirement, A, is displayed on the new relation row. After selecting document 1 and requirement A, relation successfully saved.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Create the diagram</td>
<td><em>Success</em>. Created documents, their requirements, and the relation is visible on the diagram</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 5.12:** Test results of Scenario 1 (Figure 5.1)

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Step Content</th>
<th>First Result</th>
<th>Proposed Solution</th>
<th>Second Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Repeat steps 1-6 of Scenario 1 (Figure 5.1)</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Delete one of the requirements; (A or B)</td>
<td><strong>Failed</strong>. Requirement is allowed to be deleted which accumulated other errors.</td>
<td>Prevent deletion and show a feedback.</td>
<td><em>Success</em>.</td>
</tr>
<tr>
<td>3</td>
<td>Delete one of the documents; (1 or 2)</td>
<td><strong>Failed</strong>. Document is allowed to be deleted which accumulated other errors.</td>
<td>Prevent deletion and show a feedback.</td>
<td><em>Success</em>.</td>
</tr>
<tr>
<td>4</td>
<td>Delete the relation</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Delete requirement A</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Delete document 1</td>
<td><em>Success</em>.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 5.13:** Test results of Scenario 2 (Figure 5.2)
Chapter 6

Further Research Roadmap

Defining good limits for an implementation is important. To include all the desired functionality at once is not feasible. So this thesis implementation is done with a specific domain of systems (that elicitation is performed for) in mind. Inside this domain, the implementation’s focus is further concentrated on the migration of paper-documents into digital datatypes of the new system. Even further limitations were put on the implementation: only a single description field is taken for each object, a hard-coded static database was used, a real remote web-server was not used, etc. In this chapter, how these limitations could be taken back and the implementation could be extended with these excluded features.

The selected domain of software systems was ECM systems and more specifically the elicitation process for their document digitalization process was picked as the focus. This created a comfortable subject matter for the thesis implementation. However, to be realistically usable in a complete elicitation process, the tool need to be more generalized. ReqIF supports the following atomic data types:

- **String** A unicode text string. The maximum length can be set on the Datatype.
- **Boolean** A Boolean value. No customization is possible.
- **Integer** An integer value. The maximum and minimum can be set on the Datatype.
- **Real** A real value. The maximum and minimum can be set on the Datatype, as well as the accuracy.
- **Date** A date- and timestamp value. No customization is possible.
- **Enumeration** An enumeration Datatype consists of a number of enumeration values. The AttributeDefinition determines whether the values are single value or multiple values.
Out of these atomic data types, only *String* is put to use in this implementation. As mentioned in 4, ReqIF’s SpecObjectType has a list of Attributes which gives these Attributes to a SpecObject. These Attribute objects could encapsulate any of the atomic datatypes listed above. Attributes are not used in this implementation. They provide very powerful customization possibilities of requirements to include different types of attributes. In order to generalize the tool to be used for other parts of the project domain (e.g. operational requirements), implementation of these datatypes is needed. For example, let’s say there is a document that the organization issues with 3 signatures and these signatures should be provided by 3 out of 7 people who are authorized. A requirement object that takes a number attribute would be useful here.

Another part that deserves future work is the diagramming feature. In Chapter 4, it was mentioned that reconnecting and deleting features of links were disabled. Backing these features instead of removing them from JointJS could introduce better usability. However, the difficulties experienced while extending and customizing JointJS inspires looking for another diagramming solutions in the future.
Chapter 7

Conclusion

It is important to pay attention to finding correct requirements to deliver quality software systems. RL deals with gathering and structuring these requirements. Studies show that poor requirements elicitation is an important factor in project failures. To improve the quality of RL An important problem in RL is the quality of elicited requirements. Improving quality is related to constraints such as time costs and scheduling, and organizing elicitation activities through a solid recording system. In this thesis, a tool is proposed to address these issues. First, the tool is aimed to be simple and hence, usable by nonengineer stakeholders after an introduction. Second, it creates a common environment which fosters requirement organization while elicitation is going on. The common RL techniques mostly rely on manual note-taking during elicitation activities which makes it cumbersome to structure gathered information.

The selected domain was document management systems of ECM systems. The tool consists of a server which provides persistent storage for the data and a client-side that is designed to help elicitation of requirements for migrating physical documents into an ECM system. The proposed tool uses an OMG standard, ReqIF, which provides a solid backbone. The gathered requirements are kept in ReqIF data structures. The information is gathered through a web-page UI that utilizes some JavaScript libraries; KnockoutJS, JointJS, jQuery. The system makes use of AJAX to ensure a smooth interface that does partial, fast updates.

Quality of requirements is aimed to be increased through ease-of-use, freedom of scheduling, and structured recording of requirements. The system could not be tested within a real project environment. However, tests on the UI was performed. The problems found during these tests were solved, then failed tests were repeated. In final results, all designed tests were successful. The future improvements for the tool are also presented. These improvements include the generalization of the tool; to use the versatility of ReqIF standard to its full extent so its usage can be extended towards the whole ECM and ultimately to other domains as well.
Appendix A

Implementation Code

In this appendix, selected classes of the implementation are listed.

**LISTING A.1: document.html file.** The single HTML page of the application. "data-bind" attributes work with Knockout.JS

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  <link rel="stylesheet" href="res/jointjs/joint.nojquery.css" />
  <link rel="stylesheet" href="res/bootstrap-3.1.1-dist/css/bootstrap.min.css" />
</head>
<body>
  <div class="container">
    <h1>Documents</h1>
    <table class="table">
      <thead>
        <tr>
          <th>ID</th>
          <th>Title</th>
          <th>Delete</th>
        </tr>
      </thead>
      <tbody data-bind="foreach: documents">
          <td data-bind="text: id"></td>
          <td><input type="text" style="cursor: pointer" placeholder="Enter description..." data-bind="click: $root.dummy, value: description, clickBubble: false"></td>
          <td><input type="button" value="Delete" class="btn btn-small btn-danger" data-bind="click: $root.deleteDocument.bind($data), clickBubble: false"></td>
        </tr>
      </tbody>
    </table>
    <input type="button" class="btn btn-small btn-primary" data-bind="{ click: addDocument, value: 'Add Document' }" />
    <input type="button" class="btn btn-small btn-primary" data-bind="{ click: saveDocuments, value: 'Save' }" />
    <input type="button" value="Diagram" class="btn btn-small btn-primary" data-bind="click: $root.generateDiagram" />
  </div>
</body>
</html>
```

Field Requirements of <span data-bind="text: description"></span>
### Appendix A. Appendix Title Here

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Linked Document</th>
<th>Linked Requirement</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Other Requirements**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Add Field Requirement**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```javascript
var canvaswidth = 800;
var canvasheight = 600;
var graph = new joint.dia.Graph;
var paper = new joint.dia.Paper({
  el : $('#myholder'),
  width : canvaswidth,
  height : canvasheight,
});
```

Appendix A. Appendix Title Here

57
Appendix A. Appendix Title Here

gridSize : 1,
model : graph,
perpendicularLinks : true
});

var findJointjsObject = function(arr, reqifID) {
  for (var i = 0; i < arr.length; i++) {
    if (arr[i].attributes.reqifID == reqifID)
      return arr[i];
  }
  return null;
};

var findSpec = function(docs, specId) {
  for (var i = 0; i < docs.length; i++) {
    for (var j = 0; j < docs[i].fieldRequirements().length; j++) {
      if (docs[i].fieldRequirements()[j].id() == specId)
        return docs[i].fieldRequirements()[j];
    }
    for (var k = 0; k < docs[i].otherRequirements().length; k++) {
      if (docs[i].otherRequirements()[k].id() == specId)
        return docs[i].otherRequirements()[k];
    }
  }
  return null;
};

function RootModel() {
  var self = this;
  self.rootObj = ko.observable();
  self.documents = ko.observableArray();
  self.relations = ko.observableArray();
  self.selectedDocument = ko.observable();
  self.selectedSpecObject = ko.observable();
  self.showedRelations = ko.observableArray();
  self.dummy = function() { },
  self.getDocumentById = function(id) {
    for (var i = 0; i < self.documents().length; i++) {
      if (self.documents()[i].id() == id)
        return self.documents()[i];
    }
    return null;
  },
  self.setRoot = function(root) {
    self.rootObj(new Root(root));
    self.documents.removeAll();
    for (var i = 0; i < root.documents.length; i++)
      self.documents.push(new Document(root.documents[i]));
    self.relations(
      []
    );
    for (i = 0; i < root.relations.length; i++) {
      var rel = new SpecRelation();
      rel.id(root.relations[i].id);
      rel.description = root.relations[i].description;
      rel.source = findSpec(self.documents(), root.relations[i].source);
      rel.target = findSpec(self.documents(), root.relations[i].target);
      rel.linkedFrom(rel.source());
      rel.linkedTo(rel.target());
      self.relations.push(rel);
    }
  },
  // Insert Row
  self.addDocument = function() {
    var doc = new Document();
    self.documents.push(doc);
    self.ajaxSaveDocument(doc);
  },
  self.addFieldRequirement = function() {
    var specobject = new SpecObject(self.selectedDocument().id);
    self.selectedDocument().fieldRequirements.push(specobject);
    self.ajaxSaveSpecObject(specobject, 'true');
  },
  self.addOtherRequirement = function() {
    var specobject = new SpecObject(self.selectedDocument().id);
    self.selectedDocument().otherRequirements.push(specobject);
    self.ajaxSaveSpecObject(specobject, 'false');
  },
  self.addRelation = function() {
    var rel = new SpecRelation();
    rel.source = self.selectedSpecObject();
    rel.linkedFrom(self.selectedSpecObject());
    self.showedRelations.push(rel);
    self.relations.push(rel);
    self.ajaxSaveSpecRelation(rel);
// Select Row
self.selectDocument = function(document) {
    if (self.selectedDocument() == document) {
        self.selectedDocument(null);
    } else {
        self.selectedDocument(document);
    }
},

self.selectSpecObject = function(specObject) {
    if (self.selectedSpecObject() == specObject) {
        self.selectedSpecObject(null);
        self.showedRelations([]);
    } else {
        self.selectedSpecObject(specObject);
        self.showedRelations(self.getRelatedRelations(specObject));
    }
},

self.getRelatedRelations = function(specObject) {
    var arr = [];
    for (var i = 0; i < self.relations().length; i++) {
        if (self.relations()[i].source() == specObject) {
            self.relations()[i].linkedDoc(self.getDocumentById(self.relations()[i].target().docId));
            self.relations()[i].linkedTo(self.relations()[i].target());
            self.relations()[i].linkedFrom(self.relations()[i].source());
            arr.push(self.relations()[i]);
        } else {
            if (self.relations()[i].target() == specObject) {
                self.relations()[i].linkedDoc(self.getDocumentById(self.relations()[i].source().docId));
                self.relations()[i].linkedTo(self.relations()[i].source());
                self.relations()[i].linkedFrom(self.relations()[i].target());
                arr.push(self.relations()[i]);
            }
        }
    }
    return arr;
},

// Saving Grid
self.saveDocuments = function() {
    for (var i = 0; i < self.documents().length; i++) {
        self.ajaxSaveDocument(self.documents()[i]);
    },

self.saveFieldSpecObjects = function() {
    for (var i = 0; i < self.selectedDocument().fieldRequirements().length; i++) {
        self.ajaxSaveSpecObject(self.selectedDocument().fieldRequirements()[i], 'true');
    }
},

self.saveOtherSpecObjects = function() {
    for (var i = 0; i < self.selectedDocument().otherRequirements().length; i++) {
        self.ajaxSaveSpecObject(self.selectedDocument().otherRequirements()[i], 'false');
    }
},

self.saveRelations = function() {
    for (var i = 0; i < self.showedRelations().length; i++) {
        self.ajaxSaveSpecRelation(self.showedRelations()[i]);
    }
},

// Deletion
self.deleteDocument = function(document) {
    if (document.fieldRequirements().length != 0 || document.otherRequirements().length != 0) {
        alert('Document has requirements and cannot be deleted! Delete requirements first!');
        return;
    }
    if (document.id() != null) {
        self.ajaxDeleteDocument(document);
    } else {
        self.documents.remove(document);
        alert('Document deleted! Description: ' + document.description);
    }
},

self.deleteSpecObject = function(specObject) {
    if (self.getRelatedRelations(specObject).length != 0) {
        alert('Requirement has relations and cannot be deleted! Delete relations first!');
        return;
    }
    if (specObject.id() != null) {
        self.ajaxDeleteSpecObject(self.selectedDocument(), specObject);
    } else {
        document.fieldRequirements.remove(spec);
        document.otherRequirements.remove(spec);
        alert('Requirement deleted! Description: ' + spec.description);
    }
}
Appendix A. Appendix Title Here

self.deleteRelation = function(relation) {
    if (relation.id() != null) {
        self.ajaxDeleteRelation(relation);
    } else {
        self.relations.remove(relation);
        self.showedRelations.remove(relation);
        alert('Relation deleted! Description: ' + relation.description);
    }
}

// AJAX
self.getRoot = function() {
    $.ajax(
        {url : 'ReqIFServlet?action=getRoot',
        type : 'GET',
        success : function(responseData, textStatusStr, jqXHR) {
            if (responseData.successful) {
                self.setRoot(responseData.root);
            } else {
                alert('getroot unsuccessful ' + responseData.message);
            },
            error : function(jqXHR, textStatusStr, errorStr) {
                alert(errorStr + ' error at getroot');
            }
        }
    });
}

// AJAX DOCUMENTS
self.ajaxSaveDocument = function(document) {
    $.ajax(
        {url : 'DocumentServlet',
        type : 'POST',
        contentType : 'application/json; charset=utf-8',
        dataType : 'json',
        data : ko.toJSON(document),
        data : JSON.stringify(
            {id : document.id(),
             description : document.description,
             fieldRequirements : document.fieldRequirements(),
             otherRequirements : document.otherRequirements()},
        success : function(responseData, textStatusStr, jqXHR) {
            if (responseData.successful) {
                document.id(responseData.id);
                if (responseData.saveType == 'UPDATED')
                    alert('Updated document! ID: ' + responseData.id + ", Description: " + document.description);
            } else {
                alert('Save unsuccessful! ' + responseData.message);
            },
            error : function(jqXHR, textStatusStr, errorStr) {
                alert(errorStr + ' error at ajaxSaveDocuments');
            }
        }
    });
}

self.ajaxDeleteDocument = function(document) {
    $.ajax(
        {url : 'DocumentServlet?id=' + document.id(),
        type : 'DELETE',
        success : function(responseData, textStatusStr, jqXHR) {
            if (responseData.successful) {
                self.documents.remove(document);
                alert('Document deleted! Description: ' + document.description);
            } else {
                alert('delete unsuccessful ' + responseData.message);
            },
            error : function(jqXHR, textStatusStr, errorStr) {
                alert(errorStr + ' error at deleteDocument');
            }
        }
    });
}

// AJAX SPEC OBJECTS
self.ajaxSaveSpecObject = function(specobject, isField) {
    $.ajax(
        {url : 'SpecObjectServlet?isField=' + isField,
        type : 'POST',
        contentType : 'application/json; charset=utf-8',
        dataType : 'json',
        data : ko.toJSON(specobject),
        success : function(responseData, textStatusStr, jqXHR) {
            if (responseData.successful) {
                specobject.id(responseData.id);
                if (responseData.saveType == 'UPDATED')
                    alert('Updated requirement! ID: ' + responseData.id + ", Description: " + specobject.description);
            } else {
                alert('insert unsuccessful ' + responseData.message);
            },
            error : function(jqXHR, textStatusStr, errorStr) {
                alert(errorStr + ' error at ajaxSaveSpecObjects');
            }
        }
    });
error : function(jqXHR, textStatusStr, errorStr) {
  alert(errorStr + ' ' + 'error at insertSpecObject');
});
});

self.ajaxDeleteSpecObject = function(document, spec) {
  $.ajax{
    url : 'SpecObjectServlet?id=' + spec.id() + '&docID=' + document.id(),
    type : 'DELETE',
    success : function(responseData, textStatusStr, jqXHR) {
      if (responseData.successful) {
        document.fieldRequirements.remove(spec);
        document.otherRequirements.remove(spec);
        alert('Requirement deleted! Description: ' + spec.description);
      } else {
        alert('delete unsuccessful ' + responseData.message);
      }
    },
    error : function(jqXHR, textStatusStr, errorStr) {
      alert(errorStr + ' ' + 'error at deleteSpecObject');
    };
  });

// AJAX RELATIONS
self.ajaxSaveSpecRelation = function(relation) {
  $.ajax{
    url : 'SpecRelationServlet',
    type : 'POST',
    contentType : 'application/json; charset=utf-8',
    dataType : 'json',
    data : JSON.stringify(
        {id : relation.id(),
         description : relation.description,
         source : relation.linkedFrom().id(),
         target : relation.linkedTo().id(),
         if (responseData.successful) {
          relation.id(responseData.id);
          relation.target(relation.linkedTo());
          if (responseData.saveType == 'UPDATED')
            alert("Updated relation! ID: " + responseData.id + ", Description: " + relation.description);
        } else {
          alert('insert unsuccessful ' + responseData.message);
        }
    },
    error : function(jqXHR, textStatusStr, errorStr) {
      alert(errorStr + ' ' + 'error at ajaxInsertRelation');
    };
  });

self.ajaxDeleteRelation = function(relation) {
  $.ajax{
    url : 'SpecRelationServlet?id=' + relation.id(),
    type : 'DELETE',
    success : function(responseData, textStatusStr, jqXHR) {
      if (responseData.successful) {
        self.relations.remove(relation);
        self.showedRelations.remove(relation);
        alert('Relation deleted! Description: ' + relation.description);
      } else {
        alert('delete unsuccessful ' + responseData.message);
      }
    },
    error : function(jqXHR, textStatusStr, errorStr) {
      alert(errorStr + ' ' + 'error at ajaxDeleteRelation');
    };
  });

// Diagram function
self.generateDiagram = function() {
  graph.resetCells();
  var docsJS = ko.toJS(self.documents);
  var reqs = [];
  for (var i = 0; i < docsJS.length; i++) {
    var fieldReqs = [];
    _.each(docsJS[i].fieldRequirements, function(c) {
      fieldReqs.push(new reqif.Requirement(
        reqifID : c.id,
        description : c.description
      ));
    });
    var otherReqs = [];
    _.each(docsJS[i].otherRequirements, function(c) {
      otherReqs.push(new reqif.Requirement(
        reqifID : c.id,
        description : c.description
      ));
    });
    // Diagram code
  }
var doc = new reqif.Document({
    position: {
        x: 20 + 250 * i,
        y: 50
    },
    reqifID: docsJS[i].id,
    description: docsJS[i].description,
    fieldRequirements: fieldReqs,
    otherRequirements: otherReqs
});
graph.addCell(doc);
_.each(fieldReqs, function(c) {
    graph.addCell(c);
    doc.embed(c);
});
_.each(otherReqs, function(c) {
    graph.addCell(c);
    doc.embed(c);
});
reqs = reqs.concat(fieldReqs);
reqs = reqs.concat(otherReqs);
var relsJS = ko.toJS(self.relations);
var rels = [];
for (i = 0; i < relsJS.length; i++) {
    var sourceField = findJointjsObject(reqs, relsJS[i].source.id);
    var sourceOther = findJointjsObject(reqs, relsJS[i].source.id);
    var targetField = findJointjsObject(reqs, relsJS[i].target.id);
    var targetOther = findJointjsObject(reqs, relsJS[i].target.id);
    var source = sourceField == null ? sourceOther : sourceField;
    var target = targetField == null ? targetOther : targetField;
    rels.push(new joint.dia.Link(
        source: {
            id: source.id
        },
        target: {
            id: target.id
        },
        attrs: { '.marker-source': { d: 'M 10 0 L 0 5 L 10 10 z' },
            '.marker-target': { d: 'M 10 0 L 0 5 L 10 10 z' },
            smooth: true,
            labels: {
                position: .5,
                attrs: { text: { text: relsJS[i].id, fill: 'white', 'font-family': 'sans-serif' },
                    rect: { stroke: '#3498DB', 'stroke-width': 5, rx: 1, ry: 1 }}
            }
    }));
}
var legendtext = [];
_.each(relsJS, function(c) {
    legendtext.push(c.id + ': ' + c.description);
});
_.each(rels, function(c) {
    graph.addCell(c);
});
var legend = new reqif.RelationLegend({
    position: { x: 0 , y: canvasheight-100 },
    size: { width: 240, height: 100 },
    attributes: legendtext,
});
graph.addCell(legend);
alert('Diagram generated! Please scroll down to investigate.');
});
});
}
function Root(root) {
    this.id = '';
    this.headerTitle = '';
    if (root) {
        this.id = root.id;
        this.headerTitle = root.headerTitle;
    }
}
function Document(document) {
    this.id = ko.observable();
    this.description = '';
    this.fieldRequirements = ko.observableArray();
    this.otherRequirements = ko.observableArray();
    if (document) {
        this.id(document.id);
        this.description = document.description;
        for (var i = 0; i < document.fieldRequirements.length; i++) {
            this.fieldRequirements.push(new SpecObject(document.id,
                document.fieldRequirements[i]));
        }
        for (i = 0; i < document.otherRequirements.length; i++)
            null;
            null;
L
ISTING A.3: joint.reqif.js file. Custom JointJS classes used for diagramming pur-
poses. Referenced from document-models.js (A.2).

reqif = {};
reqif.Document = joint.shapes.basic.Generic.extend({
    markup: [
        '<g class="rotatable">',
        '<g class="scalable">',
        '<rect class="uml-class-description-rect"/><rect class="uml-class-fieldRequirements-rect"/><rect class="uml-class-otherRequirements-rect"/>',
        '</g>',
        '<text class="uml-class-description-text"/>',
        '<text class="uml-class-fieldRequirements-text"/>',
        '<text class="uml-class-otherRequirements-text"/>',
        '</g>
    ].join(''),
    defaults: joint.util.deepSupplement({
        type: 'uml.Class',
        attrs: {
            rect: { 'width': 200 },
            '.uml-class-description-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#####' },
            '.uml-class-fieldRequirements-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#####' },
            '.uml-class-otherRequirements-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#####' },
            '.uml-class-description-text': {
                    'ref': '.uml-class-description-rect', 'ref-y': .5, 'ref-x': .5, 'text-anchor': 'middle', 'y-alignment': 'middle', 'font-weight': 'bold',
                    'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman' },
            '.uml-class-fieldRequirements-text': {
                    'ref': '.uml-class-fieldRequirements-rect', 'ref-y': .5, 'ref-x': .5, 'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman' },
            '.uml-class-otherRequirements-text': {
                    'ref': '.uml-class-otherRequirements-rect', 'ref-y': .5, 'ref-x': .5, 'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman' }
    },
    reqifID: 'def',
    description: [],
    fieldRequirements: [],
    otherRequirements: []
}), joint.shapes.basic.Generic.prototype.defaults,
initialize: function() {
    _.bindAll(this, 'updateRectangles');
    this.on('change:description change:fieldRequirements change:otherRequirements', function() {
        this.updateRectangles();
        this.trigger('uml-update');
    });
    this.updateRectangles();
    joint.shapes.basic.Generic.prototype.initialize.apply(this, arguments);
},
getClassName: function() {
    return this.get('description');
},
updateRectangles: function() {
    var attrs = this.get('attrs');
    var rects = [
        { type: 'description', text: this.getClassName() },
        { type: 'fieldRequirements', text: this.get('fieldRequirements') },
        { type: 'otherRequirements', text: this.get('otherRequirements') }
    ];
    var offsetY = 0;
    _.each(rects, function(rect) {
        var lines = _.isArray(rect.text) ? rect.text : [rect.text];
        var rectHeight = lines.length * 20 + 10;
        attrs['.uml-class-' + rect.type + '-text'].text = _.isArray(rect.text) ? '' : lines.join('
');
        attrs['.uml-class-' + rect.type + '-rect'].height = rectHeight;
        attrs['.uml-class-' + rect.type + '-rect'].transform = 'translate(0,' + offsetY + ')';
        offsetY += rectHeight;
    });
    var reqoffset = this.attributes.position.y + attrs['.uml-class-description-rect'].height + 3;
    for (var i = 0; i < rects[1].text.length; i++) {
        rects[1].text[i].attributes.position.x = this.attributes.position.x;
        rects[1].text[i].attributes.position.y = reqoffset;
        reqoffset += 22;
    }
    reqoffset += 3;
    for (var i = 0; i < rects[2].text.length; i++) {
        rects[2].text[i].attributes.position.x = this.attributes.position.x;
        rects[2].text[i].attributes.position.y = reqoffset;
        reqoffset += 22;
    }
    this.attributes.size.height = offsetY;
    this.attributes.size.width = attrs.rect['width'];
});
reqif.Requirement = joint.shapes.basic.Generic.extend({
    markup: [
        '<g class="rotatable">',
        '<g class="scalable">',
        '<rect class="uml-class-description-rect"/>',
        '</g>',
        '<text class="uml-class-description-text"/>',
        '</g>
    ].join(''),
    defaults: joint.util.deepSupplement({
        type: 'uml.Class',
        attrs: {
            rect: { 'width': 200 },
            '.uml-class-description-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#FFFFEE' },
            '.uml-class-description-text': { 'ref': '.uml-class-description-rect', 'ref-y': 5, 'ref-x': 5, 'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman', 'cursor': 'pointer' }
        },
        reqifID: 'def',
        description: []
    }, joint.shapes.basic.Generic.prototype.defaults),
});
initialize: function() {
    _.bindAll(this, 'updateRectangles');
    this.on('change:description', function() {
        this.updateRectangles();
        this.trigger('uml-update');
    });
    this.updateRectangles();
    joint.shapes.basic.Generic.prototype.initialize.apply(this, arguments);
},
updateRectangles: function() {
    var attr = this.get('attrs');
    var rects = [
        { type: 'description', text: this.get('description') }
    ];
    var offsetY = 0;
    _.each(rects, function(rect) {
        var lines = _.isArray(rect.text) ? rect.text : [rect.text];
        var rectHeight = lines.length * 20 + 2;
        attr['.uml-class-' + rect.type + '-text'].text = lines.join('
');
        attr['.uml-class-' + rect.type + '-rect'].height = rectHeight;
        attr['.uml-class-' + rect.type + '-rect'].transform = 'translate(0,' + offsetY + ')';
        offsetY += rectHeight;
    });
    this.attributes.size.height = offsetY;
    this.attributes.size.width = attr.rect['width'];
};
reqif.RelationLegend = joint.shapes.basic.Generic.extend({
    markup: [
        '<g class="rotatable">',
        '<g class="scalable">',
        '<rect class="uml-class-name-rect"/><rect class="uml-class-attrs-rect"/>
        '</g>',
        '<text class="uml-class-name-text"/>
        '<text class="uml-class-attrs-text"/>
        '</g>
    ].join(''),
    defaults: joint.util.deepSupplement({
        type: 'uml.Class',
        attr: {
            rect: { 'width': 200 },
            '.uml-class-name-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#ffffff' },
            '.uml-class-attrs-rect': { 'stroke': 'black', 'stroke-width': 2, 'fill': '#ffffff' },
            '.uml-class-name-text': {
                'ref': '.uml-class-name-rect', 'ref-y': 0.5, 'ref-x': 0.5, 'text-anchor': 'middle', 'y-alignment': 'middle', 'font-weight': 'bold',
                'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman'
            },
            '.uml-class-attrs-text': {
                'ref': '.uml-class-attrs-rect', 'ref-y': 5, 'ref-x': 5,
                'fill': 'black', 'font-size': 12, 'font-family': 'Times New Roman'
            }
        },
        name: 'Relation Legend',
        attributes: []
    }, joint.shapes.basic.Generic.prototype.defaults),
    initialize: function() {
        _.bindAll(this, 'updateRectangles');
        this.on('change:name change:attributes', function() {
            this.updateRectangles();
            this.trigger('uml-update');
        });
        this.updateRectangles();
        joint.shapes.basic.Generic.prototype.initialize.apply(this, arguments);
    },
    getClassName: function() {
        return this.get('name');
    }
});
updateRectangles: function() {

    var attrs = this.get('attrs');
    var rects = [
        { type: 'name', text: this.getClassName() },
        { type: 'attrs', text: this.get('attributes') }
    ];
    var offsetY = 0;
    _.each(rects, function(rect) {
        var lines = _.isArray(rect.text) ? rect.text : [rect.text];
        var rectHeight = lines.length * 20 + 2;
        attrs['.uml-class-' + rect.type + '-text'].text = lines.join('
');
        attrs['.uml-class-' + rect.type + '-rect'].height = rectHeight;
        attrs['.uml-class-' + rect.type + '-rect'].transform = 'translate(0,'+ offsetY + ')';
        offsetY += rectHeight;
    });
    this.attributes.size.height = offsetY;
    this.attributes.size.width = attrs.rect['width'];
};

LISTING A.4: SpecObjectReqIF.java file. One of the four wrapper classes presented in Chapter 4.
LISTING A.5: DocumentServlet.java file. One of the four servlet classes presented in Chapter 4.

```java
package com.ozgurkadir.servlets;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ozgurkadir.backend.DocumentRepository;
import com.ozgurkadir.backend.DocumentReqIF;
import com.ozgurkadir.servlets.SaveResponse.SaveType;
@WebServlet("/DocumentServlet")
public class DocumentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public DocumentServlet() {
        super();
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // SAVE (Insert & Update)
        RequestContext ctx = new RequestContext(request, response);
        try {
            SaveResponse resp = new SaveResponse();
            resp.setSuccessful(true);
            DocumentReqIF document = ctx.getRequestContentAs(DocumentReqIF.class);
            if (document.getID() == null) {
                DocumentRepository.insertDocument(document);
                resp.setSaveType(SaveType.INSERTED);
            } else {
                if (DocumentRepository.updateDocument(document))
                    resp.setSaveType(SaveType.UPDATED);
                else
                    resp.setSaveType(SaveType.NONE);
            }
            resp.setId(document.getID());
            ctx.writeToResponseJson(resp);
        } catch (Throwable t) {
            t.printStackTrace();
            Response resp = new Response();
            resp.setMessage(t.getMessage());
            ctx.writeToResponseJson(resp);
        }
    }
    @Override
    protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // DELETE
        RequestContext ctx = new RequestContext(request, response);
        try {
            String id = ctx.getParam("id");
            DocumentRepository.deleteDocument(id);
            Response resp = new Response();
            resp.setSuccessful(true);
            ctx.writeToResponseJson(resp);
        } catch (Throwable t) {
            t.printStackTrace();
            Response resp = new Response();
            resp.setMessage(t.getMessage());
            ctx.writeToResponseJson(resp);
        }
    }
}
```


```java
package com.ozgurkadir.servlets;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
```
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ozgurkadir.utils.Util;

public class RequestContext {
    private final HttpServletRequest request;
    private final HttpServletResponse response;
    private final InputStream requestStream;
    private final OutputStream responseStream;
    private Map<String, String> params;

    public RequestContext(HttpServletRequest request, HttpServletResponse response) throws IOException {
        this.request = request;
        this.response = response;
        this.requestStream = request.getInputStream();
        this.responseStream = response.getOutputStream();
        try {
            this.request.setCharacterEncoding("utf-8");
            this.response.setCharacterEncoding("utf-8");
        } catch (UnsupportedEncodingException e) {
        }
        this.setResponseContentType("text/plain; charset=utf-8");
    }

    public String getParam(String key) {
        if (params == null) {
            params = new HashMap<String, String>();
            String queryString = request.getQueryString();
            String[] pairs = queryString.split("&");
            for (String pair : pairs) {
                String[] keyValue = pair.split("=");
                params.put(keyValue[0], Util.Misc.decodeUrl(keyValue[1]));
            }
        }
        return params.get(key);
    }

    public <T> T getRequestContentAs(Class<T> clazz) throws IOException {
        this.setResponseContentType("application/json; charset=utf-8");
        JsonEncoder<T> encoder = new JsonEncoder<T>(clazz);
        return encoder.decode(Util.IO.readUtf8String(requestStream));
    }

    @SuppressWarnings("unchecked")
    public <T> void writeToResponseJson(T responseObject) throws IOException {
        this.setResponseContentType("application/json; charset=utf-8");
        JsonEncoder<T> encoder = new JsonEncoder<T>((Class<T>)responseObject.getClass());
        String json = encoder.encode(responseObject);
        PrintWriter out = new PrintWriter(responseStream);
        out.print(json);
        out.flush();
    }

    public void setResponseContentType(String contentTypeHeader) {
        this.response.setContentType(contentTypeHeader);
    }
}

package com.ozgurkadir.servlets;

public class Response {
    private boolean isSuccessful;
    private String message;

    public boolean isSuccessful() {
        return isSuccessful;
    }

    public void setSuccessful(boolean isSuccessful) {
        this.isSuccessful = isSuccessful;
    }

    public String getMessage() {
        return message;
    }
}

LISTING A.7: Response.java file. Wraps a response object.
public void setMessage(String message) {
    this.message = message;
}

LISTING A.8: SaveResponse.java file. Special response object for save operations.

class SaveResponse extends Response {
    protected static enum SaveType {NONE, INSERTED, UPDATED};
    private String id;
    private SaveType saveType;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public SaveType getSaveType() {
        return saveType;
    }
    public void setSaveType(SaveType type) {
        this.saveType = type;
    }
}

LISTING A.9: JsonEncoder.java file. JSON processor class that uses Jackson

class JsonEncoder<T> {
    private final static ObjectMapper mapper = new ObjectMapper();
    private final Class<T> type;
    public JsonEncoder(Class<T> type) {
        this.type = type;
    }
    public T decode(String json) {
        try {
            return mapper.readValue(json, type);
        } catch (JsonParseException e) {
            e.printStackTrace();
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    public String encode(T obj) {
        try {
            return mapper.writeValueAsString(obj);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        } return "";
    }
}
Bibliography


