Requel: A Collaborative Requirements Tool with Automated Assistance

Ronald Regan Jr.

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

June 2009

Abstract

Poor requirements have been cited as the primary cause of software project failures and cost overruns. Fredrick Brooks tells us that “The hardest single part of building a software system is deciding precisely what to build.”

What causes poor quality requirements and why is it so hard to acquiring requirements that are understandable, unambiguous, precise, and complete? One of the key problems in identifying requirements is a lack of stakeholder involvement. All stakeholders, including users, managers, and developers, need to be involved during requirements acquisition. Another problem is the high level of detail necessary to implement software. Most people don’t think about the intricacies of the tasks they perform at a conscious level.

Requel is a requirements engineering system that supports collaboration among business and technical stakeholders and provides automated assistance to validate requirements and suggest improvements. It is a Web-based application to facilitate a distributed team of users. It supports collaboration with a semi-structured discussion and negotiation mechanism. It assists users by applying natural language processing to the requirements to detect and report ambiguity and complexity. It identifies potential significant terms in the requirements to assist in build a glossary and making the requirements more understandable and consistent.

Dedication

This thesis is dedicated to my wife Theresa, for all the support and love she has given that made this dream possible.

Acknowledgments

I would like to thank my thesis director Dr. Bill Robinson for his support throughout the thesis process. His guidance helped me get to the end when I would lose focus and stray off the path.

I would also like to thank all the people out there creating great open source software that this project would not have been possible without.

Table of Contents

List of Tables

List of Equations

List of Figures

Chapter 1 Introduction

This thesis describes Requel, a collaborative tool that helps business and technical stakeholders to identify and elaborate a set of software requirements. Requel supports users with automated assistance that analyzes the requirements, identifies issues, and suggests solutions.

The primary objective of the tool is to facilitate the creation of high quality software requirements. More precisely, requirements should be:

- Understandable and unambiguous – all stakeholders will understand and agree on the expected behavior of the desired system; all terminology that may be ambiguous or unfamiliar to any of the stakeholders is clearly defined in a glossary.
- Precise and complete – there is enough explicit information in the requirements that a developer can implement them; a tester can verify that the implementation behaves correctly; and all the clients' needs are met.
- Well-organized – Relationships and dependencies between requirements elements are indicated clearly.

One of the key problems in identifying requirements is a lack of stakeholder involvement (Finkelstein, 1994, p. 4.) All stakeholders including users, managers, and developers need to be involved. Requel supports a collaborative environment for all stakeholders to participate in the process, including the following properties and features:

- Accessibility – Requel supports concurrent and distributed access through a rich Web interface in a Web browser so stakeholders can access it from anywhere.
- Process Adaptability – Requel is adaptable to different software engineering processes and activities by supporting requirements using a set of simple elements that includes goals, stories, actors, scenarios, and use-cases, but does not require using all element types.
- Extensibility and Customizability – Requel can interact with other tools through an import/export mechanism using XML and XSLT. It supports customization of documentation through custom XSLT scripts.
- Issue discussion and negotiation – Requel supports a structured process for identifying and resolving issues through discussion of various candidate solutions with multiple arguments that support or oppose the solution to assist in making decisions.

The rest of this chapter has an overview of the requirements engineering domain and how Requel supports it; a description of the natural language processing tasks that Requel uses to perform analysis; a description of similar tools in the requirements engineering space and the approaches they use to solve problems similar to those proposed here.

Software Requirements Engineering

Poor requirements have been cited as the primary cause of software project failures and cost overruns to the point of being a cliché. Fredrick Brooks said it best over

20 years ago in his article “No Silver Bullet: Essence and Accidents of Software Engineering” (Brooks, 1987)

> “The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements…”

The crux of the problem lies in the level of detail and precision necessary to produce software. Brooks (1987) asserts that “the client does not know what he wants” and that “it is really impossible for a client to specify the exact requirements” because of the detail necessary.

The field of requirements engineering has grown out of the need to identify and solve the problems that make identifying requirements so difficult. As this project is concerned with collecting software requirements it is important to understand the goals and issues of requirements engineering to see what problems the proposed tool intendeds to solve.

There are many definitions for requirements and what the requirements engineering process entails that make it a confusing topic. Furthermore there are many names given to the process, such as elicitation, gathering, defining, and so forth that add to the confusion. In the context of this thesis “requirements” means “the functionality, qualities, and constraints imposed on a system to correctly support its purpose.” Given this definition the requirements engineering process is tasked with both defining the purpose in precise terms, and identifying the functionality and qualities that meet the purpose and the constraints imposed by the environment.

When working on a complex problem decomposing it into simpler sub-problems is an effective approach. Many authors have decomposed the space of requirements engineering into various sub-areas. Finkelstein (1994) identifies seven areas of concern that reflect the basic structure in the requirements engineering process: context, groundwork, acquisition, modeling and specification, analysis, measurement, and communication and documentation. Nuseibeh and Easterbrook (2000) expand on Finkelstein's areas, combine some and extract others, although primarily focusing on the activities. Weigers (1999, pp. 38-39) organizes his best practices into seven categories: knowledge, requirements management, project management, elicitation, analysis, specification and verification.

Below are descriptions of the relevant areas of concern borrowing the concepts from Finkelstein and Weigers. For each area of concern there is a description of how Requel addresses it.

Acquisition

Acquisition is concerned with identifying the environment, stakeholders and requirements of the project (Finkelstein, 1994, pp. 3-4.) The earliest activities are concerned with understanding the problem from the point of view of the organizational and environmental setting (Mylopoulos & Castro, 2000, p. 2.) and why the system is needed (Yu 1997, p. 1.) Acquisition is a continuous and iterative process of refinement.

Stakeholders

Stakeholder analysis involves identifying the roles of the people and organizations that should be involved in the project, the individuals or organizations that fill those roles, and the responsibilities, capacities and relationships of those stakeholders (Finkelstein, 1994, p. 3.) Robinson and Volkov (1997, p. 1) extend analysis to include the stakeholders "which affect or are affected by the system" such as developers and managers. Before the stakeholders can be understood they must be identified. Sharp, Finkelstein, and Galal (1999) point out that there wasn't a lot of help in the literature for identifying the stakeholders of a project, and propose a simple approach. They include examples of roles such as end-users, managers, developers, and regulatory bodies. They also describe categorizations that have been suggested by other authors that focus on the stakeholders' relationship to the system being developed. Stakeholders may be identified as a project progresses; a project may start with a user heavy team and add technical stakeholders after a first pass to help elaborate technical details.

Beyond helping to identify stakeholders the roles of stakeholders also indicate their responsibilities in a project. Business stakeholders drive the business level and quality requirements and make decisions about the scope of the product. They include executives, managers, or staff from various disciplines like marketing, sales, services, and support. User stakeholders or surrogate users drive the user level requirements. Technical stakeholders include requirements engineers, domain experts and developers that elaborate functional requirements from the business, user and quality requirements. Finally there are supporting stakeholders that may not contribute to the requirements, but play a role in the process. Project managers organize the activities and measure progress and costs; quality assurance engineers make sure the requirements can be verified as the system is implemented.

Requel supports individual stakeholders, which are system users, as well as non-user stakeholders to represent organizations or regulatory bodies. Stakeholders may have goals that define the environmental (business, technical, and legal) concerns particular to that stakeholder. Requel doesn't support explicit roles, but it does support different permissions for authorizing user stakeholders' ability to edit, annotate and delete the requirement elements as a way to restrict activity of users.

Requirements

Requirements are identified at various levels from high level goals to low level functions, and not in any particular order. Requirements are not gathered in a strictly top down approach because "people don't think that way" (Rich & Waters, 1990, p. 200.) The identification of one use case or goal may lead to others, or a new step in one scenario may indicate the need for a corresponding step in another. To facilitate working top-down, bottom-up or middle-out, Requel supports creating requirements elements in any order. For example a team may start by creating stories based on actual events and elaborate goals and use cases that abstract the main concepts in the stories.

Different requirements acquisition methods use different entities to describe the requirements. Agile methods, such as eXtreme Programming attempt to minimize documentation and focus on user stories to define requirements. A user story is a simple narrative for the interaction of a user with the system. It is not intended to capture all the details of the interaction, but to be more of an overview of how a feature may work. Requel supports textual stories with a simple type of "success" or "exception" indicating if the story describes normal usage or a problem. Stories may have goal or actor associations and be used as “casual” use cases using Cockburn’s terminology (2000, p. 120.)

Goal oriented methods such as Eric Yu’s i* or KAOS (Knowledge Acquisition in automated Specification) model requirements in graphs of goals and their relationships, where goals may support or conflict with each other. Weigers covers a variety of models for describing requirements, including qualities and functional requirements of the form “the system shall…” Using Lamsweerdem’s definition of a goal (2001, p. 2) as “an objective the system under consideration should achieve,” Weigers qualities and functional requirements can be considered as types of goals. Requel uses goals in a general sense that includes answers to why, what and how questions concerning the system being developed. In this sense the goals include desired qualities, properties, constraints, features and functions.

Weigers (1999, p. 7) identifies three distinct levels of requirements: business, user and functional, as well as various nonfunctional or quality requirements. In this author’s opinion separating the goals into fine grained categories can be tedious for users and it doesn’t add significant value because the level of abstraction or intent of a goal is evident in the language used. For example “The system should be easy to use for non-technical users” is a soft goal that doesn’t lead to a specific function of the system, but dictates a quality of the system. On the other hand a goal like “The system needs to be accessible to users in multiple locations” dictates a mode of operation for the system as a whole. Finally, goals like “The system must support an IBIS style discussion and negotiation of issues” dictate a feature and functionality that the system must support. Not categorizing the goals removes the burden of the user having to choose a category and the impact of the user choosing the wrong category.

More important than categorizing goals is identifying the relationships between them. Requel supports identifying a directed relationship from one goal to another with a quality indicating that the source goal supports or conflicts with the target goal. For example having the goal “a user must assign a level to a goal for it to be processed” conflicts with the goal that the system should be easy to use. The relationships help users understand how the goals impact or influence each other and identify areas where tradeoffs between conflicting goals are needed.

Scenario based methods such as CREWS (Cooperative Requirements Engineering with Scenarios) use structured scenarios to define processes and agents that invoke them. Methods such as OOSE or RUP focus on use cases, which are typically composed of multiple scenarios. Scenarios in CREWS differ from use cases in that they specify only a single flow through a process. Multiple scenarios would be needed to describe a single use case with alternate and exceptional paths. There are a variety of formats for structuring use-cases in natural language. Cockburn (2000, ch. 11) covers textual use-cases from fully dressed formats with detailed scenarios for success and failure cases, to casual formats that present scenarios as narratives more like eXtreme Programming user stories. Requel supports scenario, use case, and story elements that can be used to represent scenarios and use cases of various methods. Each scenario can have multiple steps, where a step may be a single statement or a reference to another scenario. Each step has a type: pre-condition, primary, alternative, exception, and optional. Pre-conditions aren’t actually steps but represent conditions that must hold for the use case to start. The other types represent the different flows through a use case. A use case has a primary actor, a scenario, references to auxiliary actors, goals, and stories for example uses of the use case. As noted before stories have goals and actors and can be used as casual use cases.

Communication and Specification

Requirements engineering is fundamentally an exercise in communication and understanding. The customer stakeholders must communicate their needs so that the development stakeholders can understand what they truly need and determine what is feasible in the provided environment.

The requirements specification is an agreement between the customer and developers that defines in a precise way what the resulting implementation should do (Weigers 1999, p. 148). There are a variety of ways to structure the specification; both Weigers (1999, p. 153-154) and Cockburn (2000, p. 13-14) include outlines and descriptions of what should typically be included in a specification.

Given the varying documentation needs of projects, Requel supports customizable documentation generation using XSLT stylesheets. A project may have multiple stylesheets to generate documents with different levels of details and formats. It is also possible to create diagrams using a stylesheet that outputs SVG (scalable vector graphics), an XML language for generating diagrams.

Discussion and negotiation are an important aspect of collaboration. To facilitate the elaboration of the requirements all of the requirements elements support annotation with issues. A stakeholder can add an issue such as a question or problem to a requirement element to start a discussion. Users can then propose solutions to the issue and argue the merits of each with arguments supporting or conflicting with the solution. Once all the proposed solutions have been discussed one can be chosen as the resolution. Discussions can be referred to later to understand the rationale behind a requirement.

Dealing with informality is a key aspect of general knowledge acquisition characterized by domain specific terminology and ambiguity (Reubenstein, & Waters, 1991, p. 228.) A glossary defining the key terms of the project is an important tool in combating ambiguity and sharing understanding. This is particularly important when a term is used in different ways between the customer's domain and the software engineering domain.

Requel supports a glossary with terms and definitions. Terms that have the same meaning in a project may be linked together under a canonical term. It can then replace all occurrences of the subordinate terms with the canonical term to make the requirements more consistent. The automated assistance identifies terms in the text of the requirement elements that may be ambiguous and require definition. For each term the assistant adds an issue to a requirement element that the term may need to be added to the glossary with automated resolutions that add the term to the glossary and back link to the element.

Analysis

Analysis is concerned with validating the requirements to determine how correctly the requirements identify and communicate the needs of the stakeholders. Validation of requirements involves checking that requirements are:

- Accurate – the requirements are what the stakeholders want;
- Unambiguous – all stakeholders agree on what each requirement means; each term means only one thing in the context of the requirements.
- Understandable – all stakeholders know what each requirement means;
- Consistent – requirements don't contradict each other;
- Complete – all the requirements and only the requirements are documented and they are documented to a level of precision where they can be implemented and tested.

Some analysis may be automated, particularly analysis that can be modeled in rules, such as lexical analysis related to syntax or grammar. For example "I ran to the" is obviously incomplete and missing a noun, for example "store". Requel supports limited lexical analysis using natural language phrase structure and dependency parsing to identify ungrammatical sentences. Analysis that is subjective, such as determining if a requirement is really what the client wants, requires manual analysis by a stakeholder.

Accurate

Accuracy is impossible to detect and measure automatically because it is subjective (Reubenstein & Waters, 1991, p. 229.) The only way to measure accuracy is to have the customer stakeholders indicate that the requirements reflects what they want. Requel makes no attempt to analyze the accuracy of the requirements.

Unambiguous

Ambiguity can occur at different levels in natural language. Words can have different meanings and it isn't always clear which sense of a word is being used when the meanings are similar, or when there are specialized domain specific uses. The structure of sentences can also suffer from ambiguity particularly around conjunctions, called coordination ambiguity (Jurafsky & Martin, 2008, p. 433), and prepositional or adverbial phrases, called attachment ambiguity (Jurafsky & Martin, 2008, p. 432.) For example in the sentence "The man sat quietly with a bottle of wine and cigar on his table" there are multiple ambiguities. The adverb "quietly" may mean the man is being silent, that he is not moving, or that he was careful when moving into the seated position. The phrase "on the table" could refer to where the man sat, where the wine and cigar were placed or where only the cigar was placed. The pronoun "his" may refer to the man or to someone else identified in an earlier sentence. The last case is known as an anaphora reference meaning a reference to something that was introduced earlier in the text (Jurafsky & Martin, 2008, p. 696.)

Words with multiple meanings can be determined using a lexical database such as WordNet (Fellbaum 1998.) For example "bank" may be an institution for holding money, a container for holding money, or the sloped embankment of a river. Each meaning is a distinct entity in WordNet.

Ambiguity can also be caused by vagueness. For example, the word "vehicle" is more abstract than the word "automobile." The requirement "there must be parking for six vehicles" can be interpreted differently by different readers without constraints on the type or dimensions of the vehicle.

The generality or abstractness of a word can be determined using a lexical database such as WordNet where the words are organized in a hierarchy of hypernym-hyponym relations where hyponyms have a more specific meaning than the hypernyms. For example “car” is a hyponym of “vehicle.”

Requel uses the WordNet hypernym-hyponym relations to calculate an information content value using an algorithm by Seco, Veale and Hayes (2004.) The basic idea is that words higher in the hierarchy convey less information than words lower in the hierarchy, otherwise the lower words wouldn’t be needed. The algorithm, represented formally in Equation 1, calculates a value that gets smaller as the count of the hyponyms of the word increases. In the equation the function hypo(c) returns the count of all the words subsumed by the word c and the value max_{wn} represents the count of all words in WordNet with the same part of speech. The information content of each word in a sentence is compared to a threshold, by default 0.50, and words below the threshold are identified as potentially vague.

$$ic_{wn}(c) = 1 - \frac{log(hypo(c) + 1)}{log(max_{wn})}$$

Equation 1: WordNet Information Content

Using a glossary to explicitly define how words are used in a project can help disambiguate words, as long as the words are used consistently, which may be hard to detect; although Gale, Church, and Yarowsky (1992) found that typically only one sense of a word is used at a time.

> “If a polysemous word such as sentence appears two or more times in a well-written discourse, it is extremely likely that they will all share the same sense.”

Understandable

Measuring understanding is impossible without the stakeholder explicitly indicating that they do or do not understand a specific requirement. Like ambiguity, using linguistic analysis can help determine the complexity of statements in a requirement, which may be an indication of its ability to be understood. Requel uses a very simple complexity measure based on the depth of the phrase structure of a sentence. A more rigorous method using parsing and lexical information is left for future work.

Consistent use of language is important in making requirements understandable. For example, using a single term per concept helps avoid confusion. Detecting terms that are likely to refer to the same concept is difficult and shares a lot of the same issues and identifying ambiguity. Requel does support identifying terms manually or automatically, and then manually indicating that they represent the same concept. Once terms are linked as being the same concept the system can then replace the duplicate term with the preferred term so that the concept is consistently referred to throughout the requirements.

Consistent

Requirements shouldn’t contradict each other. Logical consistency can be measured from different perspectives, for example from simple logical contradictions to more subtle issues like ordering in time. Contradiction is likely to be the easiest to automatically measure by logically modeling the requirements and detecting contradictions in the logic. Given that Requel uses natural language to define the requirements, converting them to accurate logical models is not a trivial task and it is left for future work.

Complete

There are two aspects to completeness: all the requirements are known, and all the information about a particular requirement is known. In the later case with the use of semantic information and domain knowledge it may be possible to detect an incomplete requirement. This is another task that is very difficult. Requel supports some limited semantic analysis based on verb usage patterns using VerbNet (Schuler 2005), but there is not enough data to be effective and improvements are left for future work. Identifying missing requirements may be possible in a limited fashion by identifying goals that are not represented in any use cases, or by goals without relationships to other goals.

Methods and Techniques

Nuseibeh and Easterbrook (2000, p. 4) define six classes of techniques and methods: general data gathering techniques like surveys and interviews; group or team oriented techniques like focus groups and the Joint Application Design (JAD) method; prototyping; goal and scenario-based methods such as Knowledge Acquisition in automated Specification (KAOS) and Cooperative Requirements Engineering With Scenarios (CREWS); cognitive techniques adapted from psychology and knowledge engineering such as protocol analysis to understand how a user thinks about a task; and contextual or ethnographic techniques where the tasks of a process are observed while a user performs them.

Christel and Kang (1992, p. 16) organize techniques and methods into the general requirements engineering areas of information gathering, analysis and validation and then rate them on how well they solve key elicitation issues as well as on their maturity and prescriptiveness. A few of the methods they include that are not already mentioned are

Issue-Based Information System (IBIS), Controlled Requirements Expression (CORE), Soft Systems Methodology (SSM), Planning and Design Methodology (PDM) and Quality Function Deployment (QFD).

Requel is not tied to a specific method. It supports a variety of entity types that may be applied as needed by the process in use. There may be cases where the requirement elements structure does not lend itself to specific information. Requel supports adding notes to all the requirements elements to support this kind of information.

Area of Concern	**Influence on Requel**
Acquisition	Requel supports stakeholder identification and a variety of elements used to define requirements including goals, actors, stories, scenarios, and use cases. Requel allows users to create requirements elements in any order or level of detail.
Communication and Specification	Requel supports the IBIS discussion and negotiation protocol and customizable documentation using XML and XSLT.
Analysis	Requel uses natural language processing to detect potential ambiguity and complexity, and to identify potential glossary terms and actors.
Methods and Techniques	Requel is not tied to a specific method and supports agile methods based on user stories; goal based methods; and scenario or use case based methods.

Table 1: Influence of Areas of Concern

Natural Language Processing

A key feature of Requel is analyzing the requirements to identify issues and make suggestions to improve their quality. As the requirements are defined in natural language Requel must process the text of the requirements to collect lexical, grammatical and semantic information used by the analysis. This section gives an overview of the tasks that are performed to collect that information.

Sentence Segmentation

A sentence segmenter takes a string of text and breaks it into a set of sentences. This is non-trivial because there are multiple punctuation marks that can end a sentence, and the marks may be used for other functions. For example, a period "." may end a sentence, separate digits in a decimal number, or end an abbreviation. A sentence segmenter is also known as a sentence boundary detector or sentencizer.

Morphology

Most words have multiple forms such as various tenses for verbs or plurals for nouns. To simplify working with multiple form words it is convenient to identify the normalized form of a word, called the "lemma." Jurafsky and Martin define morphology as "the study of the way words are built up from smaller meaning-bearing units." (2008, p. 47.)

Many words, called regular, follow a standard pattern for generating the different forms. For example, the verb "walk" derives its Past Tense by appending "-ed" and Present Participle by appending "-ing." The knowledge that describes how the spelling of a word changes in its different forms are called orthographic rules (Jurafsky & Martin, 2008, pp. 45.) Irregular words don't follow the standard patterns. For example, the verb "be" has the forms: am, are, is, was, were, be, being, been. Luckily, the number of irregular words is fixed and small enough to store efficiently in a database.

Spelling

Spell checking is important as a precursor to lexical analysis. Words spelled incorrectly can cause problems for the parser. The wrong tag may be assigned to a word, for example marking a noun as a verb, causing the parser to structure the sentence completely wrong.

Parsing

Parsing involves splitting the sentence into syntactic elements that are then grouped in various ways. The first part of parsing is identifying the words and punctuation, known as tokenization. Then each token is tagged with its part of speech such as noun, verb, determiner, or punctuation. Syntax parsers take a sentence of words and describe the relationships between the words. There are three common types of syntactic parsers: constituent, dependency, and link.

A constituent parser, also known as a phrase structure parser, organizes the words into a phrasal tree structure with terminal components like nouns and verbs and structural components like sentences and verb, preposition, and noun phrases (Jurafsky & Martin, 2008, pp. 385-387.) Figure 1 shows an example constituent parse of the sentence "An old man drove the red car quickly" visualized with the Stanford Parser Parse Visualization Tools (available at http://ai.stanford.edu/~rion/parsing/index.html.)

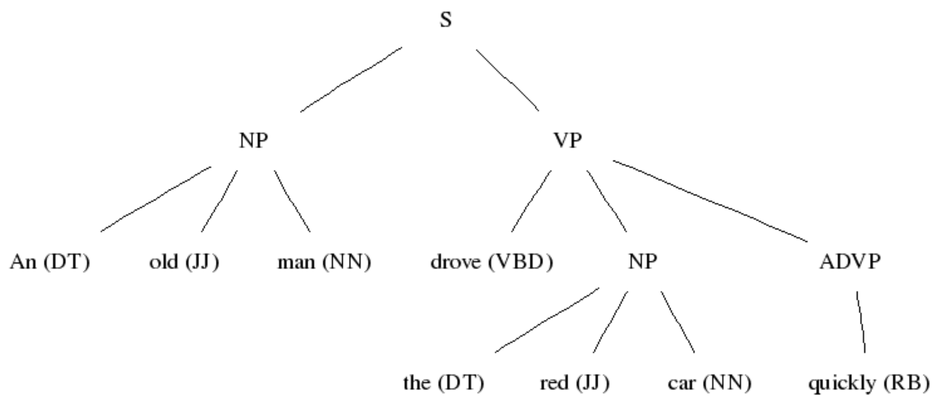


Figure 1: Example Constituent Parse

A dependency parser identifies the syntactic dependencies between the words of a sentence. For example the subject, direct object, and indirect object of a verb. Figure 2 shows a dependency parse of the same sentence as Figure 1. A link parser also returns a set of word relationships, although the relationships are derived differently and in theory are different from what a dependency parser returns.

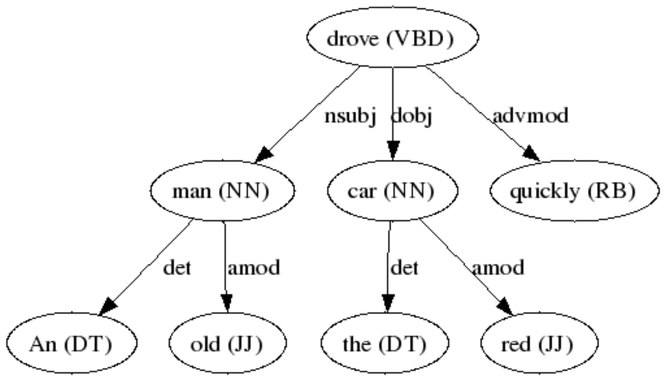


Figure 2: Example Dependency Parse

A dependency parse appears to have relationships similar to the roles assigned by a semantic role labeler, which gives the impression that it would be easy to map the dependencies to semantic roles. For example the subject becomes the agent and the indirect object becomes the patient. Unfortunately the mapping isn't that simple. Figure 3 shows the dependency parse of the original example sentence converted to a passive form.

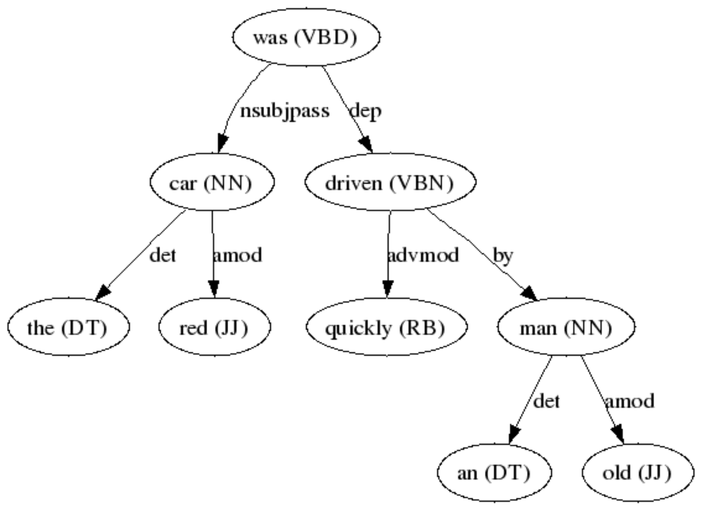


Figure 3: Example Passive Dependency Parse

Word Sense Disambiguation

Word sense disambiguation (WSD) is "the task of examining word tokens in context and determining which sense of each word is being used" (Jurafsky & Martin, 2008, p. 637.) WSD is necessary to achieve any amount of semantic understanding.

There are a few of methods for performing WSD. Co-referencing is based on the idea that the different senses of a word tend to occur more frequently with specific senses of other words. By using a corpus where all the word senses are identified the relationship between senses of different words can be learned. A benefit of this approach is that it can identify relationships between words of different parts of speech. The downside of this approach is that you need a huge corpus to collect enough data to be useful. The SemCor semantically tagged corpus identifies WordNet senses in about 350 of the Brown corpus files (Mihalcea, 1998.) Its coverage of words in WordNet that co-occur is very low.

Word similarity is based on the idea that the senses of polysemous words used together in sentences tend to be more semantically related than the other senses of the words. The semantic similarity of two words is calculated using a dictionary. There are a variety of methods for calculating word similarity.

The first algorithm based on this idea was introduced by Michael Lesk (1986.) In his algorithm the words in the definitions of all the senses of each of the words in the original sentence are compared and the senses with the most overlap were chosen. The main problem with the Lesk algorithms and its derivates is that the length of the word definitions is a key factor in its success; words with short definitions are less likely to have overlapping words (Jurafsky & Martin, 2008, p. 647.)

Another class of methods uses the hypernym-hyponym hierarchy of word relationships in a dictionary such as WordNet to calculate the similarity based on how close they are in the hierarchy (Jurafsky & Martin, 2008, p. 652-657; Simpson & Dao 2005; Banerjee 2002.) There are two predominant methods in this class; distance based methods compare the lengths of the paths between the different senses; information content based methods compare the information content of the lowest word in the hypernym-hyponym hierarchy that is an ancestor of both senses.

The main problem with semantic similarity methods is that the similarity can only be calculated for words in the same hierarchical relationship, most often the hypernym-hyponym relation, which only relates words with the same part of speech. Methods that use a probability-based information content calculation also require a sense tagged corpus to calculate the probabilities, so the issue of word coverage becomes a problem again.

An information content calculation such as the one proposed by Seco, Veale and Hayes (2004) solves the corpus problem by calculating the information content using only the hypernym-hyponym relationships.

Semantic Role Labeling

Semantic role labeling is concerned with identifying the meaning of a sentence from the meaning of the words that compose it (Jurafsky & Martin, 2008, p. 670.) A semantic role labeler assigns semantic or thematic roles to the parts of a sentence with respect to a verb. Thematic roles are a more general class of roles appropriate for many verbs. For example the statement "I eat ice cream" can be represented using the VerbNet (Schuler, 2005) thematic roles as "verb: eat, agent: I, patient: ice cream." Semantic roles tend to be verb specific. For example, using FrameNet (Ruppenhofer, Ellsworth, Petruck, Johnson, & Scheffczyk, 2006) roles on the same sentence has the form "verb: eat, eater: I, eaten: ice cream."

Figure 4 shows the semantic roles assigned to the example sentences from Figure 2 and Figure 3 generated using the semantic role labeling demo software from the Cognitive Computation Group at the Department of Computer Science, University of Illinois at Urbana-Champaign. (http://l2r.cs.uiuc.edu/~cogcomp/srl-demo.php) Note that the semantic parser assigns the "driver" role to the old man in both cases, unlike the syntactic relations assigned by the dependency parser in Figure 2 and Figure 3.

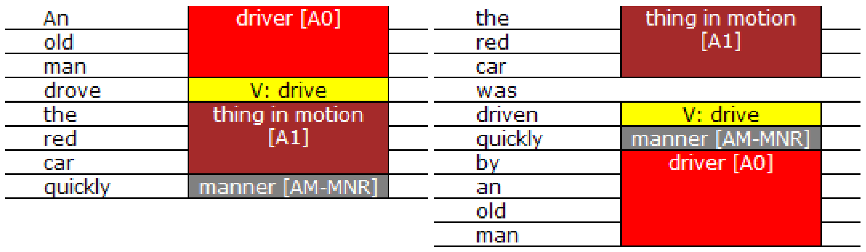


Figure 4: Semantic Role Labeling

Corpus Linguistics

A corpus isn't a process, but is important to many natural language processing tools. It is collection of texts used as an example of language use. Corpora typically contain annotated text with information related to syntax or semantics (Jurafsky & Martin, 2008, p. 130.) The Brown corpus is a million-word collection from 500 texts of different genres including newspapers, academic works, fiction, and nonfiction (Jurafsky & Martin, 2008, p. 85.) The Brown corpus is used by many natural language processing tools as training data. The SemCor corpus is a subset of the Brown corpus with additional annotations identifying the sense of the verbs, nouns, adjectives and adverbs in the sentences (Mihalcea, 1998.)

Related Tools

There are many tools specific to, or contain features for requirements engineering. The International Council on Systems Engineering (INCOSE) has a Web site for the Requirements Management Tools Survey with a matrix of tools and supported features. Ian Alexander (2007) also maintains a Web site with a list of requirements related tools and summaries of their features.

Based on a review of the tool descriptions from both sites most of the tools focus on management issues such as traceability and change management, documentation and visualization, and analysis of requirements. Borland's CaliberRM™, IBM Rational's® RequisitePro®, and Telelogic's DOORS® are the most popular commercial products that fall into the Requirements Management category.

There are very few tools in the requirements elicitation category. On Ian Alexander's site the tools "FeaturePlan" by Ryma Technology Solutions and Telelogic's "Focal Point" include features for requirements "collection" and customer involvement. The EColabor and Tuiqiao tools (Smith, 1997; Takahashi, Potts, Kumar, Ota, & Smith, 1996) work together to support elicitation and distributed collaboration.

There are a number of research projects that use processing of use cases written in a natural language like English for generating formal or semi-formal requirements models and for measuring the quality of the use cases. Tools such as Procasor and the Use Case Workshop fall into this category.

Requirements Apprentice

In the 1980's Rich and Waters (1987, 1990) among others started the Programmer's Apprentice (PA) research project at the MIT A.I. lab with the goal of understanding how expert programmers develop programs from requirements to implementation and how the tasks can be automated (Reubenstein & Waters 1991 p. 226). At that time they recognized it would be difficult to fully automate all the tasks of developing software and opted to create a system that would assist a software engineer by handling the simpler tasks and not get in the way when dealing with more complicated tasks (Reubenstein & Waters 1991 p. 226). The PA was distinct from other software engineering tools in that it was designed as an autonomous agent that cooperated with the developer through the development environment instead of being a separate tool (Rich & Waters 1990 p. 2). Rich and Waters (1987, p. 3) identify three key areas that the PA project would entail: requirements acquisition, software design, and software implementation.

The Requirements Apprentice (RA) is one of the tools conceived as part of this research (Rich & Waters, 1986, 1987, 1990; Reubenstein & Waters 1991) and is the most relevant to this thesis project. The focus of the RA is on the transformation from informal requirements into a formal specification (Rich & Waters, 1990 p. 201.) A fundamental difference between the RA and other requirements tools of the time is that it focused on the transition from informal to formal requirements rather than the validation of requirements already in a formal representation (Reubenstein & Waters 1991 p. 227.) The authors noted that creating a formal specification was a difficult task that had been avoided by previous research. The RA is not intended to interact with end-users directly, but as a tool used by an analyst. This is mainly to avoid natural language processing issues that would arise from having to communicate with "naive" end-users in free-form text (Reubenstein & Waters 1991 p. 227.)

Requel borrows the idea that the assistant makes suggestions to the user instead of just acting on the input particularly because the assistant is processing natural language in free-form text and bound to encounter language it cannot fully process or processes incorrectly. The assistant never makes changes to the requirements without a user approving them. Users can ignore bad suggestions made by the assistant.

The RA uses a layered processing approach, which Requel also uses. Requel starts with simple processing like spell checking and syntax parsing and works its way to complex processing like word sense disambiguation and semantic role labeling. Requel makes suggestions based on the level of processing it was successfully able to perform. For example, if the assistant was able to assign semantic roles to the text of a scenario step, then it can determine which text represents the actor and if that actor matches an actor defined in the requirements.

Requirements Assistant for Telecommunication Services (RATS)

The RATS tool was developed by Armin Eberlein as part of his PhD thesis on telecom service requirements (Eberlein, 1997.) It is conceptually similar to the Requirements Apprentice in that it is an expert system designed to assist a requirements engineer in creating a requirements specification using a knowledge base of telecom service domain models. RATS provides two type of assistance; passive assistance that validates the requirements against the domain knowledge and active assistance that gives suggestions on what the requirements engineer should do next based on the RATS method. The domain knowledge is implemented in Telos, a frame based logic system, using the ConceptBase knowledge base.

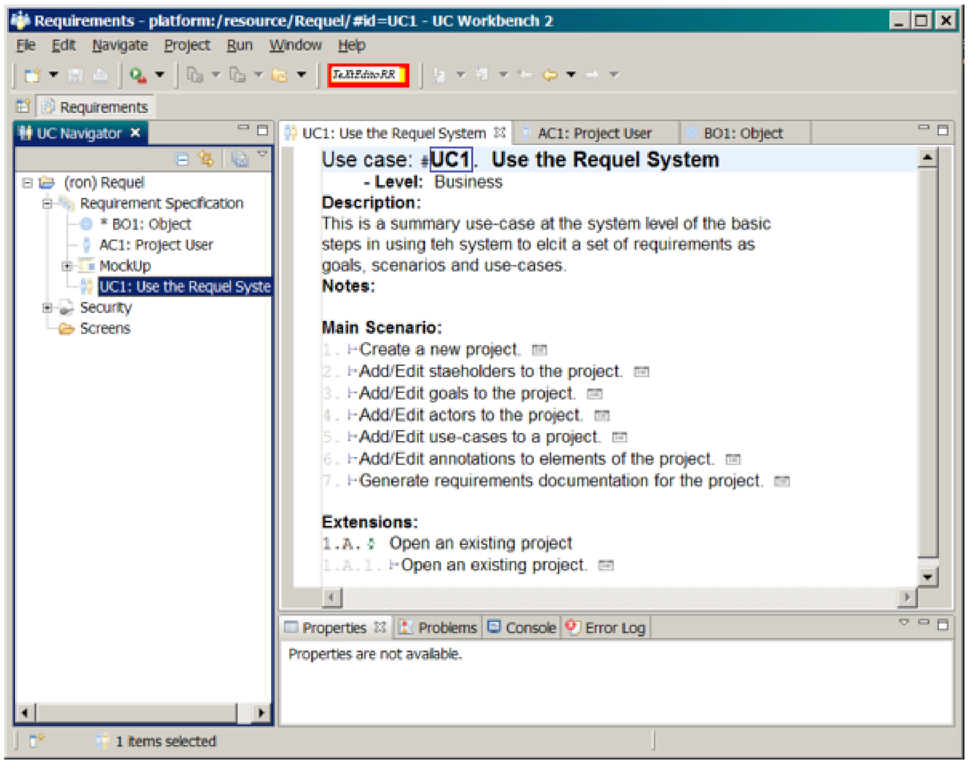
The original scope of this thesis included many ideas from the RATS system, in particular using the domain knowledge to suggest next steps, but to quote Eric Temple Bell "Time makes fools of us all."

UC Workbench

The UC Workbench is a tool for requirements analysts to create and validate use-cases for well formed-ness based on patterns of good quality use-cases (Ciemniewska, Jurkiewicz, Olek, & Nawrocki, 2007.) It uses Natural Language Processing (NLP) tools (specifically the Stanford dependency parser) to detect defects or “bad smells”, a term the authors borrow from Kent Beck, defined as a surface indication of a deeper problem. In NLP terms “surface” indicates syntactic or grammatical information versus semantic information.

The authors codify the best practices described by Cockburn (2000), and Adolph, Bramble, and Cockburn (2002) into rules used to validate use cases at the specification level, use case level, and scenario step level. At the specification level they try to detect use-cases that have the same behavior in steps, only varying in the data objects being manipulated, trying to weed out duplicates. At the use-case level they validate the name, number of steps, and complexity. At the step level they validate the grammar, sentence complexity and detect missing actors.

The UC Workbench is implemented as an Eclipse Rich Client Platform (RCP) application. It has an interesting interface for editing use-cases that is similar to a WYSIWYG interface used by most modern word processors.


Requirements - platform:/resource/Requel/#id=UC1 - UC Workbench 2
File Edit Navigate Project Run Window Help
Requirements
UC Navigator
(ron) Requel
Requirement Specification
* BO1: Object
AC1: Project User
MockUp
UC1: Use the Requel Syste
Security
Screens
UC1: Use the Requel System
AC1: Project User
BO1: Object
Properties
Problems
Console
Error Log
1 items selected
Use case: #UC1. Use the Requel System
- Level: Business
Description:
This is a summary use-case at the system level of the basic steps in using teh system to elcit a set of requirements as goals, scenarios and use-cases.
Notes:

Main Scenario:
1. Create a new project.
2. Add/Edit staeholders to the project.
3. Add/Edit goals to the project.
4. Add/Edit actors to the project.
5. Add/Edit use-cases to a project.
6. Add/Edit annotations to elements of the project.
7. Generate requirements documentation for the project.

Extensions:
1.A. Open an existing project
1.A.1. Open an existing project.

Properties are not available.

Figure 5: UC Workbench

A nice feature of the UC Workbench is that it can generate a Web-based form of the requirements document that stakeholders can navigate and review.

Requel is conceptually very similar to the UC Workbench, although the UC Workbench is intended for requirements engineers to use to create use cases and not as a collaborative tool for all stakeholders to identify and elaborate a set of requirements. The UC Workbench takes a more modest approach to natural language processing. The use cases are defined in a controlled subset of English grammar called FUSE (Nawrocki and Olek, 2005.) It expects the use case steps to have a simple structure with a subject, verb, direct object and prepositional phrase. The actor is always the subject of the sentence.

Procasor

The Procasor project is concerned with converting use cases in natural language to a more formal behavioral specification called Pro-Cases (Mencl 2004.) The project is led by Vladimir Mencl at the Charles University in Prague, Czech Republic.

Procasor is similar to the UC Workbench in that it uses NLP to analyze use-cases at the step level, although it differs in how the analysis is done. It uses the Collins constituent parser to create a phrase structure tree and then analyses the structure. A limitation of using the phrase structure is that the tool requires sentences to be restricted to the form: subject - verb - direct object - preposition - indirect object (Mencl 2004.) More recently the team is working to reduce this limitation and support steps with leading conditions and compound sentences (Dražan & Mencl 2007.)

gIBIS

The gIBIS tool is not related to requirements engineering directly, but is a standalone tool that implements the IBIS method of discussion and negotiation. The goals of the tool include keeping track of the decision making process history, and having a collaborative environment for discussions. The authors of the tool make some suggestions on enhancements to the IBIS method (Conklin & Begeman, 1988.)

Requel models its discussion and negotiation features after the IBIS method with issues, positions and arguments. Requel also supports the "null" position concept, which basically indicates ignoring the issue; and picking a position as the resolution based on suggested improvements by the authors.

EColabor and Tuiqiao

Tuiqiao (Chinese for ‘elaboration’) is a single-user hypertext tool that implements the Inquiry Cycle method (Potts, Takahashi, & Anton 1994) for requirements analysis. It allows a user to manage requirements as text and scenarios, and add annotations such as questions, answers and reasons as part of a discussion about the requirements.

The interesting aspect of this tool relevant to this thesis is the collaboration through discussions as question and answer annotations and traceability of the annotations to give rationale to the requirements, which Requel adopts and mixes with the IBIS concepts. Other aspects that fell out of scope are the versioning of individual elements of the requirements document.

Feature Comparison

Table 2 summarizes and compares the key features of the related tools described above.

	Requel	UC Workbench	Procasor	Requirements Apprentice	RATS	EColabor & Tuiqiao	gIBIS
Collaborative Environment	x					x	x
Discussion & Annotations	x					x	x
Natural Language Processing	x	x	x				
Logical Modeling of Requirements				x	x		
Informal Requirements Specification	x	x				x	
Formal Requirements Specification			x	x	x		

Table 2: Related Tools Feature Comparison

Summary

This chapter described the key aspects of requirements engineering and how Requel addresses the related issues.

For acquisition Requel supports stakeholder identification and a variety of elements used to define requirements including goals, actors, stories, scenarios, and use cases. For communication and specification Requel supports the IBIS discussion and negotiation protocol and customizable documentation using XML and XSLT. For analysis Requel uses natural language processing to detect potential ambiguity and complexity, and to identify potential glossary terms.

Requel borrows design concepts and features from prior requirements and collaboration tools. Such as the layered processing and assistant making suggestion ideas used in the Requirements Apprentice and RATS tools; the annotation and discussion concepts from the EColabor and gIBIS tools; and the natural language analysis used in the UC Workbench and Procasor tools.

Organization of this Document

The rest of this document is organized as follows. Chapter 2 defines the requirements of the Requel system as a set of stories, goals, actors and use-cases. Chapter 3 describes the design of the Requel system including the architecture, external tools used, and implementation details. Chapter 4 is a comprehensive user and installation guide. Chapter 5 concludes the document with what was learned and future work.

Chapter 2 System Requirements

This chapter gives a summary of the requirements for Requel as a set of goals, actors and use-cases with stories and scenarios as an example or snapshot of what the requirements may look like being collected by the tool.

Goals

Goals include desired properties, constraints, features and functions that the system must support. Goals are primarily textual with associations between them. Goals have a name, which is a short one line description of the goal, an optional text body which is a long description of the goal, and relationships to other goals that it supports or conflicts with. To help keep Requel simple goals don’t have an explicit type or level. Below are a few select goals of the tool.

Goal	Description, Relationships, and Notes
Easy to use	The tool must be useable by both technical and non-technical stakeholders. It must be simple for users to find and add information to the requirements. • Issue – This goal is vague and not easily testable.
Improve communication between stakeholders	Requirements are fundamentally about communication and shared understanding. The customer stakeholders must describe what they need with a high level of detail so that developer stakeholders can implement it. This is a difficult task. The tool must support features that allow a dialog between the technical and non-technical stakeholders to elaborate the details so that all the stakeholders understand what will be built.
Support discussion of requirements	The system supports an IBIS style of discussion and negotiation of issues. Issues represent problems or questions about elements of the requirements. Issues have one or more positions that are possible solutions. Positions have zero or more arguments that support or conflict with the position as the solution to the issue. • Supports "Improve communication between stakeholders."
Don't impose a process	The system is useable with various requirements acquisition processes. Users can use any process that models the requirements in the available element types • Conflicts with "Easy to use."
Don't impose top-down or bottom-up gathering of requirements	Users are able to work at different levels and in different directions from abstract to concrete and vice-versa. For example a user can start with a functional requirement or scenario and then define the features, goals and stories that explain why a functional requirement or scenario exists. After a goal or story is defined the user may go back and modify a functional requirement based on ideas or discussions about the goals and add new ones. • Supports "Easy to use." • Supports "Don't impose a process"
Remote access	The system supports users in a distributed environment. Users can access the system from remote sites as easily as if they were working locally.

Multi-user support	The system supports multiple users working concurrently on the same or different projects. If two users are working on the same data, changes by one user will not be inadvertently over written by another user. Users will see changes by other users as they happen.
Requirements in natural language	The system supports stories, goals, use cases and scenarios defined in natural language. • Supports "Easy to use." • Supports "Improve communication between stakeholders."
Automated assistance	The system assists users by analyzing the requirements to identify issues and make suggestions for improving the requirements. The analysis identifies potential glossary terms, actors and domain objects from all of the requirements elements. • Supports "Easy to use." • Issue – What types of "issues" does the analysis identify?

Table 3: Goals of the Requel System

Actors

Actors represent the users of the system, both human and other systems. Actors have a name, optional description, and goals to help refine the needs and constraints specific to this actor. There are two types of users of Requel: administrators and project stakeholders. The system has an assistant that is a pseudo-actor, it's part of the system, but acts like a user of the system.

Administrator

Administrator manages things like user accounts and system configuration.

Goals

- Create new users of the tool that will work on projects and/or be administrators.
- Add or change the system roles of users and their permissions. For example give a project user administrator privileges or authorization to create new projects.
- Edit users to change personal information or reset their passwords.

Automated Assistant

The assistant is part of the system, but acts like a user of the system adding issues and positions to the requirements in a project based on its analysis.

Goals

- Assist users by adding issues to point out potential problems or enhancements to the requirements.

Project User

Project users use the system to create requirements and may be members of one or more projects in the system as stakeholders. Some project users may create new projects; others will only be able to work on projects when they are added by other users. Project users have project specific permissions for editing different types of requirements elements.

Goals

- Create and edit requirements on one or more projects.
- Discuss requirements by adding problems, questions and notes.
- Review open issues and add comments.
- Generate a requirements document.

Table 4 gives examples of the types of project users/stakeholders and the people that fill those roles. The people represent an example team working on a fictitious accounting system project.

Stakeholder Role	**Description**	**Example Personas**
Customer/Consumer (Business stakeholders, End-users)	These are users that are part of the customer organization that will use or have an interest in the system being specified. These users primarily define the "what and why" goals of the software and review and refine the use cases, although sometimes create them outright. Examples are end users and business management or customer proxies that act on behalf of end users or business management.	**Theresa** is an accountant for Bailey Pet Supply tasked with specifying the requirements for integrating the purchase order system with the accounting system using the system. Theresa has never used the requirements tool before and has no projects. Theresa will be an end-user of the resulting system. **Buddy** is the controller of Bailey Pet Supply; he is the sponsoring manager of the project with interests in, but is not a direct user of the resulting system.
Supplier/Developer	These users are part of the team that will deliver the software requirements of the system being specified and possibly the software. These users primarily review and discuss goals and use-cases to help refine them, estimate effort and feasibility.	**Ron** is a software developer contracted by Bailey Pet Supply to create the new purchase order system. Ron has not used the tool but is familiar with requirements engineering.
Analyst/ Requirements Engineer	The analyst actor represents the requirements analyst. This user is most likely part of the development team. This user works with the customer to review and refine goals and create use-cases.	**Jason** is a requirements engineer tasked with elaborating the technical requirements for the system. Jason has lots of experience with the tool.
Project Manager	A project manager is a project user whose main objective is managing the team of project users for one or more projects in the system. A project manager may act on behalf of another stakeholder to ask or answer questions and make comments. The PM may be on the consumer	**Rich** is the project manager for the accounting system project assigned to answer questions that other team members have posted and to ask questions about requirements entered for the project.

	or supplier team.	
Authority (non-user role)	Some application domains are subject to rules by consortiums or government agencies. These agencies typically will not be represented by a stakeholder user in the system.	**FASB** is the Financial Accounting Standards Board that sets rules on accounting practices.

Table 4: Stakeholder Roles, with Examples

Use Cases

Use cases represent functional requirements of the system in terms of how a user interacts with the system. A use case has a name; an optional description; a complex scenario with alternate and exceptional paths; a set of goals that represent what the primary actor is trying to accomplish with the use case; a set of example stories including successful and exceptional cases. This section defines some of the use cases that Requel supports in varying levels of detail in the same way that it would be used to define requirements.

Create or edit a user

An administrator creates users; assigns roles such as administrator or project user; and grants permission for project users, such as the permission to create projects. Existing users can be edited to change any personal information and change roles and permissions.

Primary Actor: Administrator

<u>Goals</u>

- Create new users of the tool that will work on projects and/or be administrators.
- Add or change the system roles of users and their permissions. For example give a project user administrator privileges or authorization to create new projects.
- Edit users to change personal information or reset their passwords.

Example Success Story

Eric gets an email from Rich requesting a user account for him. Rich needs to use the system as a project user and be able to create projects. Eric logs into the system, and because he is an administrator the system gives him the option to create new users. Eric chooses to create a new user for Rich and the system displays an interface for entering a name, email address, phone number, username, password and the option to grant administrator and project user roles, and permission to create projects. Eric enters “rich” for the username and “rich123” as a temporary password. He adds the project user role and grants Rich permission to create new projects. Eric sends Rich an email with his username and password and tells rich to change his password when he first logs in.

Example Success Story

Eric gets a call from Ron that he forgot his password and needs it reset. Eric logs into the system and chooses “ron” from the list of users. In the user editing interface Eric enters a new password and saves it. Eric tells Ron to use “ron123” for his password and to change it when he next logs in.

Login to the system

All users must be authenticated by the system with a username and password to use it.

Primary Actor: Any User

Example Success Story

Theresa connects to the requirements tool via a Web browser from a link in an email message sent by Dave the system administrator. The system doesn't detect an existing user session and presents the login screen. Theresa enters her username and password supplied in the email. The system verifies the username and password combination is correct and displays the user interface for project users containing the "Purchase Order System" project that Theresa is assigned to.

Example Exception Story

Ron connects to the requirements tool via a Web browser. The system doesn't detect an existing user session and presents the login screen. Ron enters his username and an incorrect password. The system displays an error message that the username and password combination do not match a valid user. Ron calls Eric the I.T. guy and asks to reset his password.

Scenario

1. The user accesses the system through a Web browser.
2. The system determines the user doesn't have an active session and displays a login interface prompting the user for a username and password.
3. The user enters his or her username and passwords and submits them back to the system.
4. The system verifies the user supplied a correct username and password combination and displays the appropriate interface to the user based on his or her roles.
    a. (exception) The system informs the user the username and password combination are not valid and redisplays the login interface.

Create a new project

A project is a container for managing a set of requirements for a particular system. A project has a name; a description that can be used as an overview or scope; and a customer. The project name plus customer name must be unique in the system. Only authorized users can create projects.

Primary Actor: Project User

Example Success Story

Rich logs in to the system and it displays a list of his active projects and the option to create a new project. Rich chooses to create a new project and the system displays the "New Project" interface. Rich enters "Purchase Order System" for the project name, "Bailey Pet Supply" for the customer name, and leaves the description blank. The system creates the project and adds it to Rich's list of active projects.

Example Success Story

Rich logs in to the system and it displays a list of his active projects and the option to create a new project. Rich chooses to create a new project and the system displays the "New Project" interface. Rich enters "Accounting System" for the project name, "Bailey Pet Supply" for the customer name, and leaves the description blank. The system detects that a project with the name "Accounting System" already exists for the customer "Bailey Pet Supply" and displays a message that a project with that name already exists. Rich changes the name to "Purchase Order System". The system creates the project and adds it to Rich's list of active projects.

Scenario

1. (Pre-Condition) The user is authorized to create a new project.
2. The user chooses to create a new project.
3. The system displays the "New Project" interface with inputs for a name, description and customer. The customer input allows entering a new customer name or selecting an existing customer.

4. The user enters a project name, customer name and optionally a description.
    a. (alternative) The user selects an existing customer.
5. The system verifies the project name plus customer name doesn't already exist in the system, creates the project and adds it to the users list of current projects.
    a. (exception) The system determines the combination of project name plus customer name is not unique and displays a message that the project name must be unique for a customer.
    b. The user changes the name of the project to a name not already in use.
        i. (alternative) The user changes the name of the customer to another customer name.

Create or edit a stakeholder

A project will have one or more stakeholders. Stakeholders may be users of the system or non-users that represent authorities such as a government regulatory agency. User stakeholders are granted permissions to edit various requirement elements and may have goals that represent that user's interests. Non-user stakeholders will have goals that represent concerns or rules dictated by that authority.

Primary Actor: Project User

Example Success Story

After Rich creates the "Purchase Order System" project he needs to add Theresa as a stakeholder so she can add user requirements to the project. Rich selects the stakeholders interface for the project and chooses to add a new stakeholder. The system displays the "New Stakeholder" interface that includes a selector for existing system users and a list of permissions to grant to the user. Rich selects Theresa's username and grants permission for her to create actors, stories, goals, and annotate any requirement element. The system creates the stakeholder and adds it to the stakeholders for the project. The next time Theresa logs in she sees the "Purchase Order System" in her list of current projects.

Example Success Story

Rich needs to make sure that the “Purchase Order System” meets the rules set by the Financial Accounting Standards Board. Rich decides that to distinguish these requirements from the user goals he will create a non-user stakeholder and add the rules as goals of that stakeholder. Rich selects the stakeholders interface for the project and chooses to add a new stakeholder. The system displays the “New Stakeholder” interface that includes an input field for entering the name of a non-user stakeholder. Rich enters “Financial Accounting Standards Board” for the name and saves. The system creates the stakeholder and adds it to the project. Rich edits the stakeholder and chooses the option to create new goals to add to the stakeholder. As Rich adds goals they appear on the stakeholder interface in the goal section. On the goal interface the stakeholder appears in the “referenced by” section.

Example Exception Story

Theresa wants to add her boss Buddy to the system so he can review the status of the project and make decisions related to the business rules the system should honor. Theresa logs into the system and selects the stakeholders from the “Purchase Order System” project. The system displays the list of existing stakeholders but does not give her the option to add a new stakeholder because Rich didn’t grant her permission to create stakeholders.

Scenario

1. (Pre-Condition) The user is authorized to edit stakeholders in the current project.
2. The user chooses to create a new stakeholder.
    a. (alternative) the user chooses to edit an existing stakeholder.

3. The system displays the edit stakeholder interface with fields for creating or editing user and non-user stakeholders.
4. The user selects an existing system user and chooses various permissions to grant to the user for the different requirements elements.
    a. (alternative) The user enters a name for a non-user stakeholder.
5. The system verifies that a stakeholder doesn't already exist for the user in the project, creates it, and adds it to the project.
    a. (exception) The system detects that the user is already a stakeholder in the project and displays a message to the user that the selected user is already a stakeholder in the project.
    b. (alternative) The system verifies that a non-user stakeholder doesn't already exist for the supplied name in the project, creates it, and adds it to the project.
        i. (exception) The system detects that a non-user stakeholder already exists in the project for the supplied name and displays a message to the user that the name is in use.

Create or edit a goal

Goals represent desired qualities, constraints, features and functions of the system. Goals have a name; optional detailed description; and a set of relationships to other goals indicating a goal supports or conflicts with another goal.

Primary Actor: Project User

Example Success Story

Rich needs to add goals to the "Financial Accounting Standards Board" (FASB) stakeholder to indicate accounting practices the system must adhere to. Rich navigates to the FASB stakeholder and chooses to add a new goal. The system displays the "New Goal" interface with fields for a goal name and detailed description. Rich enters "Support FASB 133: Accounting for Derivative Instruments and Hedging Activities" for the goal name and the details of what the ruling requires in the description and saves it. The system creates the goal and adds it to the project and the FASB stakeholder's set of goals, and returns Rich to the stakeholder edit interface. The system starts analysis of the goal in the background.

Scenario

1. (pre-condition) The user is authorized to edit goals in the current project.
2. The user chooses to create a new goal.
   a. (alternative) the user chooses to edit an existing goal.
3. The system displays the edit goal interface with fields for the goal name and optional detailed description.
4. The user enters the name and detailed description.
5. The system verifies the goal name is unique to the project, creates the goal, adds it to the project, and starts analysis of the goal.
   a. (exception) The system determines the goal name is already in use by another goal and displays a message to the user that the goal name is in use and must be changed.

Add or edit a relationship between two goals

The system supports relationships between goals such as one goal supporting or conflicting with another goal.

Primary Actor: Project User

Example Success Story

Theresa creates a new goal in the project "The system provides help to users based on experience." After creating the goal the system displays the edit goal interface with the option to add relationships from this goal to other goals. She decides that the new goal supports the goal "The system is easy to use for new users" and chooses to add a relationship between the two goals. The system opens the goal relationship editor with the new goal in the "from" field and a list of all the other goals in the "to" field. Theresa searches through the list and finds the goal she wants and selects it. She then selects "supports" from the list of available relation types and saves. The relation appears in the new goal's table of relations to other goals.

Create or edit a use case

Use cases define the behavior of the system. A use case is a composite structure with a name, description, scenario and references to actors, goals, and stories. The name of the use case typically implies the primary goal of the primary actor, for example "create a use case."

Primary Actor: Project User

Example Success Story

After a meeting with Theresa, Jason wants to create a use case to describe the process that she follows to add a purchase order to the system. Jason logs into the requirements tool and selects the "Purchase Order Integration" project. He selects the use cases of the project and because he is authorized to add use cases the system gives him the option to add a new one. Jason chooses to add a new use case and the system prompts him for a name, description, and primary actor. Jason enters "Create a purchase order" for the name and a short description of what a purchase order is for the description. Jason scans the list of actors and chooses "Purchase Clerk". In the scenario section of the use case editor Jason sees the options to add a step or add a scenario, and he chooses to add a step. The system adds input fields to select the type of step; an empty text field for the description; the option to delete the step; and the option to add a sub-step. Jason adds a few steps and saves the use case.

Scenario

1. (Pre-Condition) The user is authorized to edit use cases in the current project.
2. The user chooses to create a new use case.
    a. (alternative) the user chooses to edit an existing use case.
3. The system displays the edit use case interface with fields for the use case name, description, primary actor, and scenario.
4. The user enters a name and description, and selects the primary actor from a list of actors in the project.
5. (optional) The user edits the scenario (see the scenario use case.)
6. The system verifies the use case name is unique to the project, creates it, adds it to the project, and starts analyzing it.
    a. (exception) The system determines the use case name is already in use by another use case and displays a message to the user that the name is in use by another user case and must be changed.

Create or edit a scenario

Scenarios are semi-structured elements describing the interaction of an actor with the system. They are typically represented as a dialog between the actor and the system where the actor does something, the system reacts, the actor does something else, etc. Scenarios are composed of an ordered list of steps where a step may be a single action or a reference to another scenario with its own set of steps. Both steps and scenarios have a type: primary, pre-condition, alternative, exception, and optional. Where primary means it is part of the primary flow of an interaction; alternative means it is an alternative but successful interaction; exception indicates a problem or error condition; and optional means it is a successful interaction, but one that is not required to take place. The pre-condition type isn’t really a step, but indicates a condition that must be met for the scenario to start or continue.

Primary Actor: Project User

Example Success Story

Ron needs to create a use case describing how users will log into the system and what they see when they login. Ron logs into the requirements tool and chooses the use cases under the “Purchase Order Integration” project. Ron chooses to add a new use case and the system displays the use case interface. After filling in the use case details Ron starts working on the scenario. He chooses to add a step and the system displays an input to select the type and enter text. Ron enters “The system displays a login screen with a username and password.” He then chooses to add another step and enters “The user enters a username and password and submits.” Finally he adds a step that “the system verifies the username and password are valid and displays the main page.”

Ron realizes that he should start the scenario with a user action. The “Purchase Order Integration” system will be Web-based so Ron adds a new step “The user connects to the purchase order system with a Web browser.” The new step is at the bottom so Ron moves it to the top of the scenario and the system reorders the other steps below it.

Example Success Story

As Ron reviews the login use case for the “Purchase Order Integration” project he realizes that there isn’t any way of handling the case of a user that forgets their password. Ron adds a story describing what should happen and then edits the main scenario. The system lists the existing steps. Ron chooses the exception step where the system determines the username and password don’t match an expected pair and changes it to include that the system displays a “forgot password” option to the redisplayed login interface. Ron then chooses to add an alternative sub-step to the exception step that the user invokes the “forgot password” action. He then adds a series of steps describe the process for recovering from a forgotten password.

<u>Scenario</u>

1. (Pre-Condition) The user is authorized to edit scenarios in the current project.
2. The user chooses to create a new scenario.
    a. (alternative) the user chooses to edit an existing scenario.
    b. (alternative) the user chooses to create or edit a use case.
3. The system displays the edit scenario interface with fields for the name, type and sub-steps.
4. The user enters the name and chooses a type.
5. (optional) The user edits the steps of the scenario
    a. (optional) The user adds a new step.
        i. The system adds input components for the type and text after the last step in the scenario.
    b. (optional) The user adds an existing scenario as a step.
        i. The system displays a list of the existing scenarios.
        ii. The user selects a scenario from the list.
        iii. The system adds the scenario as the last step in the original scenario.
    c. (optional) The user moves a step from its current position to a new position in the scenario.
        i. The system inserts the step in the new location moving the original step in that location and all steps below that location down by one location.
    d. (optional) The user removes a step from the scenario.
        i. The system removes the step from the list and moves all steps that followed the removed step up by one location.
6. The system verifies the scenario name is unique to the project, creates it, adds it to the project, and starts analyzing it.
    a. (exception) The system determines the scenario name is already in use by another scenario and displays a message to the user that the name is in use and must be changed.

Annotate a requirements element with a note or issue

The system supports adding annotations such as notes that don't support discussion or issues that support discussion and resolution. Annotations can be added directly to a project, or to elements of the project such as a goals, use cases, actors, scenarios, or steps in a scenario. Notes only have a text description; issues have a description, zero or more positions, and may have a "resolved by" position.

Primary Actor: Project User

Example Success Story

As Theresa defines the goal for reporting quarterly reports in the "Purchase Order Integration" project, she is not sure if old versions of the reports need to be kept for historical purposes. Theresa chooses to add an issue to the goal and the system displays the "New Issue" interface with a field to enter the text of the issue and an input to select if the issue needs to be resolved. In the text field Theresa enters "Does the system need to keep old versions of the report?" and saves. The system creates the issue and adds it to the goal. The issue also shows up in the open issues interface.

Example Success Story

Ron adds a goal to the "Purchase Order Integration" project with the name "Users can saerch for purchase orders by customer name" and saves it. The system invokes the assistant to analyze the goal. The assistant finds the word "saerch" that appears to be spelled incorrectly or is a word the assistant doesn't know. The assistant adds an issue to the goal that the word "saerch" in the goal name is unknown to the assistant and may be a spelling error. The assistant adds positions to the issue to ignore the word, add the word to the dictionary, and with various words known to the assistant that are spelled similarly to the unknown word, such as “search.”

Add a position to an issue

Positions represent potential solutions to an issue. A position has a text description and a set of arguments.

Primary Actor: Project User

Example Success Story

Buddy is reviewing the open issues of the “Purchase Order Integration” system and finds the issue “Does the system need to keep old versions of the report?” that Theresa added. Buddy opens the issue and sees no positions. Buddy chooses to add a position and the system opens the “New Position” interface with inputs for entering the position text and adding arguments. Buddy enters “Yes, we may need to send out old versions of reports. If a purchase order is changed that impacts how the report looks, we need a copy of the original one” and saves the position. The position appears on the issue editor in the list of positions.

Example Success Story

Jason is reviewing the open issues of the “Purchase Order Integration” system and sees the position and argument that Buddy added to the issue “Does the system need to keep old versions of the report?” Jason edits the issue and adds a new position with the text “Save old versions of reports outside the system” and saves. The new position appears on the issue in the list of positions.

Add an argument to a position

Arguments are reasons why a particular position of an issue should or should not be chosen as the solution to an issue. Arguments have text description and a support level with values from “strongly against” to “strongly for.” Users can examine the arguments for the different positions to help make a decision of which position is most appropriate.

Primary Actor: Project User

Example Success Story

After Buddy creates a position for the issue “Does the system need to keep old versions of the report?”, he edits the position and chooses to add an argument. The system displays the “New Argument” interface with inputs for entering the argument text and selecting a support level. Buddy enters the text “This is required by the standard accounting practices”, chooses “Strongly For” as the support level, and saves. Buddy then goes and edits the position “Save old versions of reports outside the system” that Jason added and adds an argument with the text “It is easier to find documents if the system keeps track of them”, chooses “For” as the support level, and saves.

Example Success Story

Jason is reviewing the open issues of the “Purchase Order Integration” system and sees the position and argument that Buddy added to the issue “Does the system need to keep old versions of the report?” Jason edits the position and adds a new argument with the text “Saving the old versions of the report will add three days to the schedule”, selects “Neutral” for the support level, and saves.

Select a Position as the resolution to an issue

Issues that are marked as “must be resolved” remain in the open issues list until a position is chosen as the resolution.

Primary Actor: Project User

Example Success Story

After a stakeholder meeting where it was decided that the system will keep track of old versions of reports, Rich finds the issue “Does the system need to keep old versions of the report?” in the project open issues interface and edits it. In the issue editor Rich chooses to resolve the issue using the position that Buddy entered. The issue editor closes and the issue is removed from the open issues interface. Rich opens the original goal of the issue and sees that the issue is marked as resolved by Rich with the text of the position used to resolve it.

Example Success Story

Ron is reviewing the open issues of the “Purchase Order Integration” and sees an issue with the text “The phrase ‘purchase order’ is a potential glossary term, actor, or domain object/property” assigned to a goal and two stories. Ron chooses to edit the issue and sees three issues: “Ignore this phrase.”, “Add ‘purchase order’ as an actor to the project”, and “Add ‘purchase order’ to the project glossary.” Ron resolves the issue with the position to add the text to the project glossary. The issue is closed and removed from the open issues interface. Ron opens the project glossary and sees the glossary term ‘purchase order’ and chooses to edit it. In the glossary term editor Ron sees the goal and two stories as “referrers” to the term. Ron opens one of the stories and sees the term listed under the terms referred to by the story.

Add or edit a term in the project glossary

Each project has a glossary of “significant” terms used in the text of the requirements elements. A term has the text of the term, such as an abbreviation, acronym, word, or phrase; a definition; and optionally a canonical term, which is the preferred term if there are multiple terms for the same concept.

Example Success Story

As Rich works on the requirements he determines that entering “Financial Accounting Standards Board” all over the place is tedious and makes the text longer. He decides that using the acronym FASB will be easier to type and read, but may not be known to non-accounting stakeholders so he should put it in the glossary. Rich selects the project glossary and chooses to add a new term. The system displays the “New Term” interface with inputs for the term, its description, and an option to select the canonical term. Rich enters “FASB” for the term and “The Financial Accounting Standards Board sets rules on accounting practices.” For the description and leaves the canonical term empty. The system creates the term and starts analysis that looks for uses of “FASB” in the text of requirements and creates a reference back to the term in the glossary.

Rich then thinks that “Financial Accounting Standards Board” is already used in a bunch of places and he should change them for consistency. Instead of manually looking for all the places where the text occurs and editing each to replace it with “FASB”, he creates another term, this time putting “Financial Accounting Standards Board” for the term and selecting “FASB” as the canonical term. The system creates the term and starts analysis to find all uses of it.

After the analysis completes, Rich selects the option to replace the term "Financial Accounting Standards Board" with the canonical term "FASB". The system updates the text of all the requirements elements that refer to the original term and replaces the text "Financial Accounting Standards Board" with "FASB".

Example Success Story

Ron goes back to the "login scenario" he created earlier and sees the assistant added a bunch of issues related to terms. For example, "The text 'a web browser' is a potential glossary term." Ron opens the issue and sees two positions: "do nothing", and "Add 'a web browser' to the project glossary." Ron chooses the "resolve" action for the second option. Back in the edit scenario interface the issue appears as resolved and in the list of referred terms "a web browser" appears. Ron chooses to edit the term and the system displays the "Edit Term" interface. He sees the "login scenario" in the referring entities list. Ron enters a description and saves the term. The system updates the term, but because Ron didn't change the term text, it doesn't start analysis to find references to the term in the other requirements, which happened when the term was first created.

Primary Actor: Project User

Goals

- Define terminology in an unambiguous way so that all stakeholders can understand it.

Link two glossary entries

Glossary entries may be cross-referenced to indicate synonyms or aliases. One term will be identified as the "canonical" term, which is the preferred term for the concept. For example users may refer to "the system" in the requirements using the terms “the tool”, and “the application.” These terms can be edited in the glossary to set “the system” as the canonical term. The system can then automatically replace the use of the alternate terms in the requirements’ text with the canonical term.

Primary Actor: Project User

Goals

- Identify the preferred term in the glossary for synonymous terms.
- Make the requirements more consistent by referring to a concept with a single term.

Example Success Story

As Ron reviews the glossary he notices that there are entries “The system” and “The application.” Ron reviews how the terms are used by looking at the requirement elements referenced by the terms and sees that the term “the system” is used mostly in the goals and scenarios, and the term “The application” is used mostly in the stories. Ron decides the term “The system” is preferred term and edits the term “The application” to make the term “The system” the canonical term and chooses the option to replace the term “The application” with the canonical term. The system updates the text of all the requirements elements that refer to the original term and replaces the text “The application” with the text “The system.”

Generate a document for a project

Users can generate project documentation by executing a report generator. Report generators hold an XSLT that is applied to an XML version of the requirements to generate a document. Each project can have multiple generators for different types of documents or reports.

Primary Actor: Project User

Example Success Story

[TODO]

Chapter 3 Design and Implementation

This chapter covers the implementation of Requel from the high level architecture to the component level details including design patterns and the technologies used.

Architecture

This section describes the architecture of Requel in terms of the patterns used and the rationale behind the choices made.

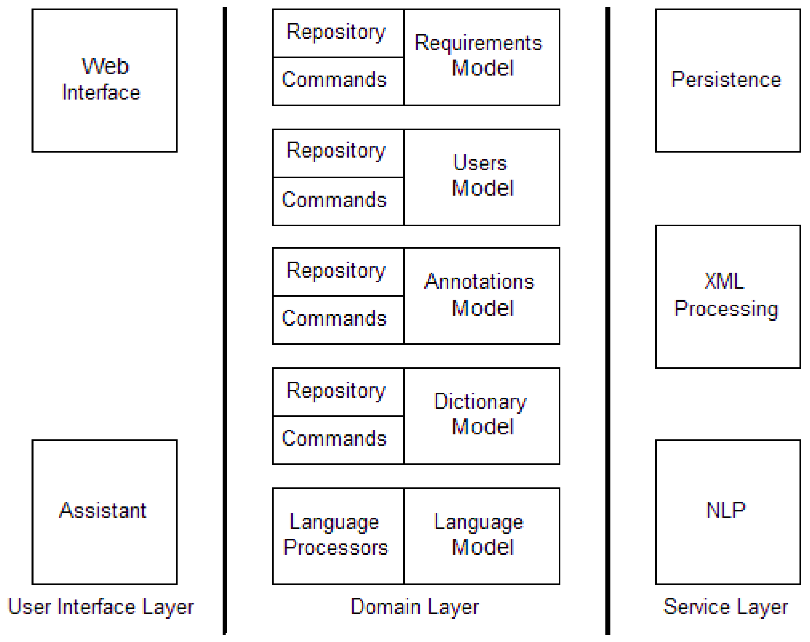


Figure 6: Architecture Layers

At the highest level of abstraction Requel uses a layered architecture (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 31-51) with three layers: user interface, domain, and services as shown in Figure 6. The layered architecture was chosen primarily to insulate the core functionality from problems with the technology choices in the user interface and supporting services such as persistence.

User Interface Layer

This section describes the two components in the user interface layer: the Web interface for human users to access the system, and the assistant that analyzes the requirements.

Web Interface Component

The Web interface is structured primarily using the document-view variant of the model-view-controller architectural pattern (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 125-143.) The model is embodied in the interface to the domain layer and the objects that represent requirements and users. The screens and panels for displaying and manipulating the requirements represent the combined view-controller aspect of the pattern.

This pattern was chosen primarily for the separation of concerns between the user interface and the underlying domain model. There were multiple user interface frameworks under consideration and there was the potential that the selected one may have to be swapped out. Using this pattern mitigated the risk to only impacting the user interface.

Table 5 lists the consequences of the MVC architecture pattern from Pattern-Oriented Software Architecture (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 141-143) relevant to the Requel system and how it impacted the implementation.

Consequence	**Impact**
Multiple views of the same model.	The interconnectedness of the domain model requires that objects appear in different views in different ways. For example a goal appears in the goal navigator, goal selector, cross-reference tables for most of the other requirement elements, cross-reference tables for annotations and in an editor.
Synchronized views.	The objects in the domain model are highly intertwined with references between them so that one object may appear in multiple views such as navigation screens and cross-reference tables on the editors of other objects. When an object is updated, all the views of the object are updated immediately resulting in a good user experience.
Framework potential.	The Requel system uses the Echo2 framework, which is MVC based. As part of the Requel system a high-level panel and event framework was created on top of the Echo2 framework that made the development of the different screens easier.
Increased complexity.	While the panel and event framework increased productivity during development of the requirements specific panels, the framework itself is very complex and it took significant effort getting the panel management and event dispatching functionality working properly.
Potential for excessive number of updates.	This isn’t a significant issue for Requel as there typically aren’t that many open views at a time and propagation of changes only occur when a user initiates an action that persists a change, such as a save, add, remove, or delete.
Intimate connection between view and controller.	At the panel level this is very true. Each panel is a combination of the view and controller. At the component level this is mostly true as well. For example the tree component has a tree model and the table component has a table model.
Close coupling of views and controllers to a model.	This is true at the panel level, for example adding, changing or removing a property on the Goal interface will most likely require a change to the views and controllers.

Table 5: Consequences of the MVC Pattern

Assistant Component

The automated assistant behaves just like the human users of the system; it reviews the requirements and creates notes and issues as needed. Given its behavior it makes sense to have the assistant interact with the domain layer the same way the Web interface interacts with it.

The assistant component follows the Whole-Part pattern (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 225-242) where the assistant is composed of many assistant parts that support a specific type of analysis. The assistant uses a Façade (Gamma, Helm, Johnson, and Vlissides, 1995, pp. 185-193) to orchestrate the actions of the assistant parts. Figure 7 shows the whole-part relationship between the assistant façade and the specialized assistants. The project, use case, and scenario assistants are themselves composites that use other assistants to analyze the components of the corresponding requirement elements.

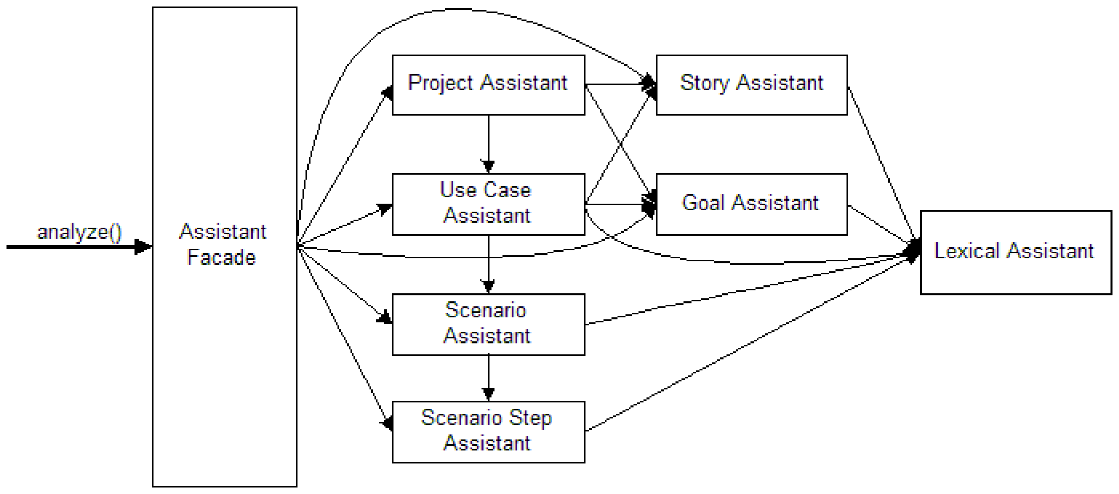


Figure 7: Assistant Architecture

Domain Layer

The domain layer is concerned with the requirements and supporting models and the actions that manipulate them. The users, requirements, and annotations components depicted in Figure 6 are an attempt to modularize the domain into logical areas of concern. The dictionary and language components are used exclusively by the assistant to insulate it from the underlying natural language processing tools and lexical data.

Domain Oriented Components

The requirements, users, annotations, and dictionary components share a similar internal structure with domain objects, a repository for finding specific objects, and a set of commands for manipulating the objects. The architecture of these components is based on the Domain Model, Repository, and Command Processor patterns.

The Domain Model pattern (Fowler, 2002, pp. 116-124) is the primary pattern dictating the structure of this layer. The requirements are modeled using the objects and relationships identified in the problem domain model. This pattern was chosen to support the rich user interface and automated analysis. The requirements elements have a lot of interdependencies that would be difficult to manage using a database record oriented approach. The downside to this approach is the complexity of the data management having to map the objects to and from the database, which is mitigated by the use of an object-relational mapping interface like the Java Persistence API.

The Repository pattern (Fowler, 2002, pp. 322-327) is used for hiding the details of the underlying persistence and querying mechanism. Each component has a repository interface for accessing the objects in that component.

The Command Processor pattern (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 277-290) is used to structure the actions on the object model into discrete units and provide a gateway between the user interface and domain layer. The primary concern for using this pattern is to isolate transactions from the user interface layer. The user interface layer components create one or more commands and pass them to the domain layer through the command processor; the commands are processed in a transaction and returned to the user interface layer components back through the command processor.

Language Component

The language component is concerned with processing natural language into constituent parts and adding lexical and semantic information to convey the meaning of the original text in a form suitable for analysis. There are a variety of processes and tools that perform different aspects of the final solution from parsing, to word sense disambiguation, to semantic role labeling. The processes are independent, but some processes rely on the information identified by other processes. For example word sense disambiguation relies on the words and parts of speech identified by the parser.

The Blackboard pattern (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, pp. 71-290) has been used by various artificial intelligence applications, including language processing. It is ideal for the situation where different tools word independently to solve different parts of a problem. Figure 8 shows the architecture of the language component structured using the blackboard pattern.

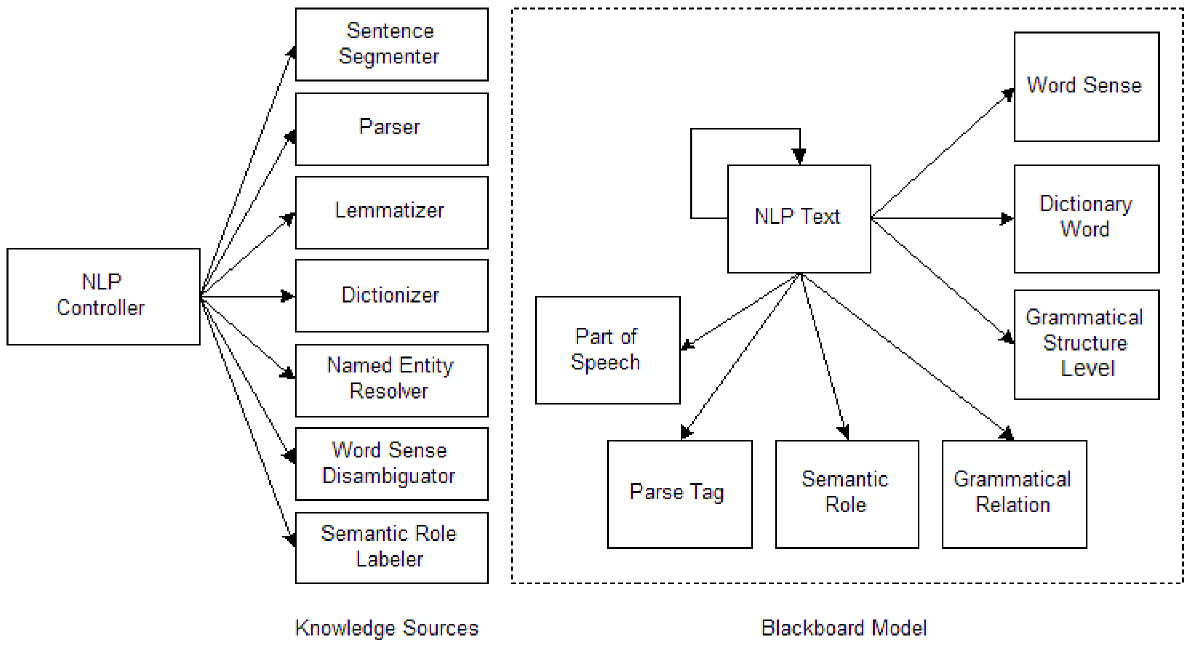


Figure 8: Language Component Architecture

The blackboard model represents the text being processed in a standardized form that is read and written to by the different knowledge sources. The NLP Text component is the primary data structure used to represent text from the paragraph level to the word level. The other components represent different pieces of knowledge attached to the NLP Text elements. For example the Part of Speech component identifies the part of speech of a word. The Parse Tag element identifies the constituent type of words, phrases, clauses, and sentences.

The knowledge sources represent the different processes applied to the text. Some are adapters to external tools that translate between the blackboard model and the tool specific representation. For example, different parsers use different sets of tags, typically depending on the corpus used to train or test them. The parser component will have to translate the tags to the ones used in the blackboard model.

The NLP Controller coordinates the execution of the knowledge sources. One of the consequences of the blackboard pattern is a lack of support for parallelism (Buschmann, Meunier, Rohnert, Sommerlad, and Stal, 1996, p. 94.) The controller must ensure access by the knowledge sources to the blackboard model is synchronized.

Service Layer

The service layer contains the components integrated with Requel to support the use cases implemented in the domain layer. The primary components in this layer are for natural language processing; XML processing used for import; export and document generation; and persistence of the domain models. Given that the service components are for the most part unrelated external tools, there is no specification for the architecture of them.

Problem Domain Model

This section defines the domain objects identified from the requirements and implemented in the domain layer components.

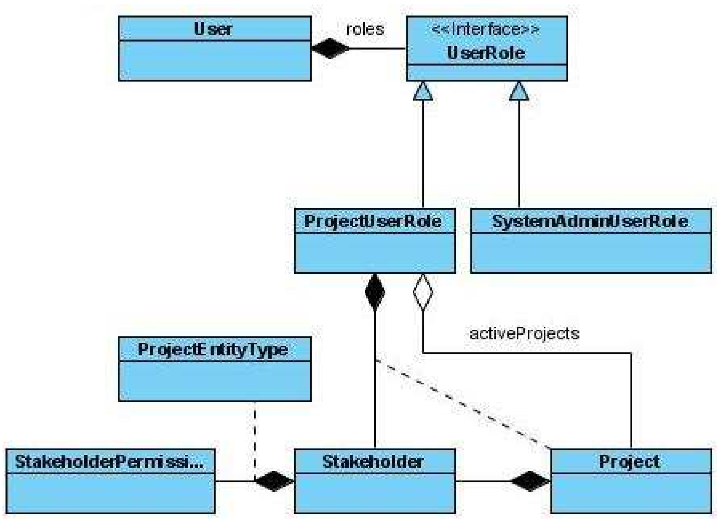


Figure 9: Users, Roles and Stakeholders

User

A user represents the users of the Requel system including project users and system administrators. It is conceivable that a user may have multiple roles in the system, such as system administrator and project user.

Username	A system-wide unique text identifier for the user.
Password	A string used to authenticate a user. This must be kept secret.
Name	The user’s real name useful for identifying the user in documents or reports generated by the system.
Email Address	The user’s email address.
Phone Number	The users primary contact number. This may be included in generated documents or presented to other users of the system for contacting this user.
User Roles	The system level roles that the user has permission to use, such as project user, and system administrator.

Table 6 User Properties

Project User Role

The project user role represents a user's general project information and permissions. This is different from a user's project-specific "stakeholder" information, such as their role, team and permissions on a specific project.

User	The owning user. This is needed because the role contains user specific information.
Active Projects	A set of projects that this user has available.
User Role Permissions	A project user may have permission to create new projects.

Table 7 Project User Role Properties

System Admin User Role

The admin user role represents the system admin actor from the use cases. A user with this role has authority to create and edit other users. The role has no properties.

Stakeholder

Stakeholders are entities such as a person or organization with an interest and possible impact on a particular project. Stakeholders have goals that a project should satisfy. A stakeholder is specific to a project. All users of the system are stakeholders, although not all stakeholders are users. The role a stakeholder plays and their goals will vary from project to project.

User	If this is a user stakeholder, this references that user.
Name	If this is a non-user stakeholder, the name of the stakeholder.
Project	The project associated to the stakeholder.
Goals	Goals representing the interests of the stakeholder.
Permissions	Permissions the user has for accessing the requirement elements.

Table 8 Stakeholder Properties

Stakeholder Permission

Stakeholders are assigned operation level permissions such as grant, edit, or delete for the different types of requirement elements.

Entity Type	The type of requirement element the permission applies to. For example goal, story, actor, etc.
Permission Type	Grant, edit, or delete.

Table 9 Stakeholder Permission Properties

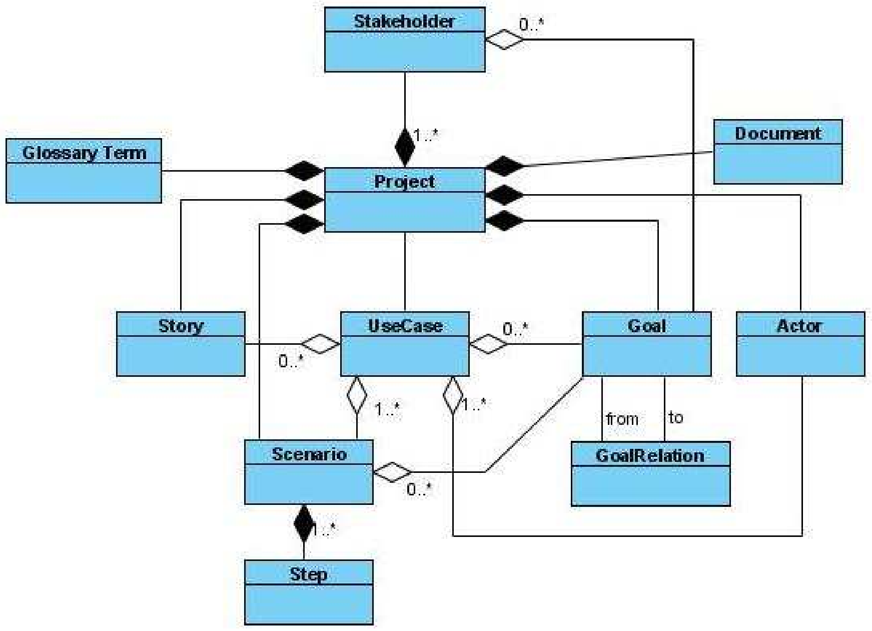


Figure 10: Project Elements

Project

A Project is a container for all the objects that make up a set of requirements and the ancillary objects created while eliciting the requirements, such as glossary terms, notes, and issues.

Name	A customer unique readable identifier for the project.
Customer	The organization the requirements are for. This is mainly for organizing projects.
Description	A summary or overview of the project.
Stakeholders	The stakeholders of the project
Goals	The goals of the project.
Stories	The stories of the project.
Actors	The actors of the project.
Scenarios	The scenarios of the project.
Annotations	Notes and issues of the project not specific to other elements of the project.
Glossary Terms	A glossary of significant terms of the project.
Documents	The XSLT stylesheets for generating different documents.

Table 10 Project Properties

Goal

Goals include desired properties, constraints, features and functions. Goals are primarily textual with associations between them.

Name	A goal has a short project unique name to make it easy find in a search and as a label in documents for cross-referencing.
Text	The text of the goal.
Relationships	The relationships this goal has with other goals.

Table 11 Goal Properties

Goal Relation

Goals may have relationships with other goals. Each relationship is directional from one goal to another. A relationship will indicate that a goal either supports or conflicts with the target goal.

From Goal	The goal that has the relationship with the "To Goal."
To Goal	The goal that is the target of the relationship.
Relation Type	Specifies if the "From Goal" supports or conflicts with the "To Goal."

Table 12 Goal Relation Properties

Story

Stories are descriptions of interactions between users and the system. They may describe desired interactions or how the system handles error conditions.

Name	A story has a short project unique name to make it easy find in a search and as a label in documents for cross-referencing.
Text	The text of the story.
Type	The type indicates if the story represents a successful or exception interaction.
Actors	A story my list the actors represented in the text.

Table 13 Story Properties

Actor

The actors represent the roles that users and other entities external to the system, such as other systems, play in the interactions that the system supports.

Name	An actor has a project unique name.
Description	A description of the role the actor plays in the system.
Goals	The goals the actor has in the system separate from the interaction specific goals.

Table 14 Actor Properties

Scenario

A scenario defines a sequence of steps used to complete an interaction with the system, or a common partial interaction. A scenario may be successful (meets all the goals) or unsuccessful (meets less than all the goals.)

Name	A scenario has a short project unique name to make it easy find in a search and as a label in documents for cross-referencing.
Use Cases	The use case(s) that use this scenario, which may be empty if scenario is exclusively used as a common step in other scenarios
Scenarios	The scenario(s) that use this scenario as a sub-step, which may be empty if this scenario is a top level scenario defining a complete interaction.

Table 15 Scenario Properties

Step

A Project is a container for the stakeholders and requirements.

Text	The text describing the activity of this step.
Type	The type of step, primary, optional, exception etc.

Table 16 Step Properties

Use Case

Use cases define a specific behavior of the system through actors, goals, stories and a scenario.

Name	A use case has a short project unique name to make it easy find in a search and as a label in documents for cross-referencing.
Description	A summary or overview of the use case.
Primary Actor	The main actor in the use case.
Goals	The goals that the actors are attempting to accomplish by doing the use case.
Stories	Stories used to augment the use case with simple narratives.
Auxiliary Actors	Additional actors the system may need to interact with to complete the use case.
Scenario	The scenario of the use case. The scenario may contain multiple paths that are successful or exceptional.

Table 17 Use Case Properties

Glossary Term

A glossary term defines a single term or concept. Terms may be cross-referenced such that two terms are identified as referring to the same concept, such as “the system” and “the application.”

Name	A project unique string representing the term. It may be a single word or a phrase.
Definition	A text definition of the term.
Canonical Term	If two (or more) terms represent a single concept; the subordinate terms will reference the primary or preferred term. A term can only be subordinate to a single term so that the meaning will be unambiguous.

Table 18 Glossary Term Properties

Document

Documents represent the types of documentation that can be generated for the project. Each project can have specialized documentation.

Name	A project unique name identifying the type of document that will be generated.
Generator	The XSLT script that generates the document from the project elements.

Table 19 Document Properties

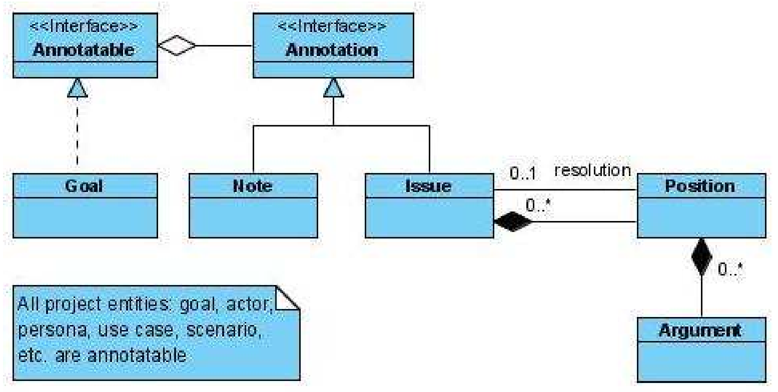


Figure 11: Annotation Objects

Note

A note is a simple annotation used to provide extra information to requirement elements. Notes don't have any discussion or resolution elements.

Text	The text of the note.
Annotated Entities	The requirement elements that refer to the note.

Table 20: Note Properties

Issue

An issue is an annotation that supports discussion through positions and arguments.

Text	The text of the issue.
Annotated Entities	The requirement elements that refer to the issue.
Positions	The potential solutions to the issue.
Resolution	The position that resolves the issue, if it is resolved.

Table 21: Issue Properties

Position

Positions represent the potential solutions to issues. They can be supported by a set of arguments to help identify the best solution.

Text	The text of the position.
Arguments	A set of arguments that argue for or against the position.
Issues	The set of issues that use this position.

Table 22: Position Properties

Argument

A position may have multiple arguments supporting or contradicting it. Collectively the arguments for all the positions should help the stakeholders agree on the deciding position.

Text	The text of the argument.
Position	The position the argument supports.
Support Rank	A rank from strongly conflicts to strongly supports to give arguments a relative weight.

Table 23: Argument Properties

External Components

This section describes the external tools and frameworks that Requel uses.

Spring Framework

The Spring Framework is an alternative or auxiliary container to the Java Enterprise Edition (JEE) platform. At its core is the Inversion of Control (IoC) container that creates and connects application objects together (Johnson, 2007, pp. 17-19.) This concept is also known as “Dependency Injection” a term coined by Martin Fowler that describes the process where the objects needed by another object to perform its service are given to that object instead of it asking for the objects it needs (Fowler, 2004.)

The biggest benefit of using the Spring Framework is in the configuration and plumbing of the application. It supports configuration through annotations on the classes and configuration files. Requel uses a mix of annotation and file based component mapping that makes configuring the system simple.

The Command Handler configuration is a good example of the power of dependency injection and the Spring Framework. During testing it was discovered that users working on the requirements could cause the analysis, running concurrently in the background, to fail due to database locking. Updating all the analysis code to catch locking exceptions and restart commands would make the code complicated and fragile. By creating a command handler decorator that intercepted the locking exceptions and re-executed the command none of the commands or code that used the commands had to change. The only change was writing the decorator and changing the configuration of the command handler to use the new class. The Spring Framework injected the new handler in all the places where the original handler was used.

> "One of the central tenets of the Spring Framework is that of non-invasiveness; this is the idea that you should not be forced to introduce framework-specific classes and interfaces into your business/domain model." (Johnson, 2007, pp. 125.)

One of the main reasons the Spring Framework was chosen is that it doesn't require programming to an API to use it. The application code can focus on the problem domain and only a small amount of code is required to get the configured objects from the Spring Framework. Direct access to the Spring Framework's container API is limited to only a few places: the main application servlet, the database initialization listener, and in the command factories via a helper class that encapsulates the bean creation strategy.

The Spring Framework offers many other services; those used in this project include transaction management, Aspect Oriented Programming (AOP), and thread pooling.

Java Persistence API and Hibernate

Object-Relational Mapping is a technique for combining the benefits of object-oriented programming and relational databases. Differences in representation of object and relational models and features of object and relational systems add complexity to this mapping. Hibernate is an open-source ORM tool that helps bridge the gap between the object and relational models. Hibernate supports the Java Persistence API, an official Java standard for managing object-relational mapping.

Hibernate is primarily concerned with transparent persistence of Java objects without the need for writing JDBC or SQL code to save and load them. Transparent means that it has minimal impact on the implementation of the application. Hibernate is designed to work with POJOs (plain old java objects.) Hibernate doesn't require the classes of the objects that will be persisted to the database to implement specific interfaces. Although you may need to include the java.io.Serializable interface and implement hashCode() and equals() methods to get associations working properly.

One of the most powerful and at times frustrating features of Hibernate is the saving or loading of objects through reachability. What this means is that Hibernate traces through all the associations of an updated object to find other objects that have been changed and saves them as well. This was very useful for project importing where JAXB would create Java objects for all the requirements elements of a project and by making the project object persistent Hibernate would save all the objects.

Another key feature of Hibernate is lazy loading, which means that when an object is loaded the objects it is associated with don't have to be loaded. Instead Hibernate creates a proxy representing the associated object and loads it from the database when it is accessed. This is extremely efficient especially in an application such as Requel that has a highly connected model. Without lazy loading all of the objects in a project would get loaded any time one of them was needed.

A problem with lazy loading is that the proxies standing in for the unloaded objects require an active session to load the data from the database. In stateless Web applications where pages are generated and all the objects released each time lazy loading doesn't cause problems. However, because Requel uses the Echo2 framework, which is not stateless, and the domain objects are used as the model objects in the user interface, lazy loading became a huge problem. In Requel's architecture the user interface and domain are in separate layers. As soon as an object was passed from the domain layer to the user interface layer it lost its connection to Hibernate and the proxies no longer had a session to load the data. Hibernate allows attaching a disconnected object to another session for loading lazy objects, but that would require having the user interface connect the object to Hibernate each time it needed to read properties making the code complicated and fragile. Another problem was that even when the objects had the data already loaded the data becomes out dated and different user interface components would end up with different versions of the objects.

Requel's solution is to add a proxy around each persistent object that caches the objects data, periodically refresh the cached objects data depending on how old the cache is, and trap accesses to Hibernate lazy loaded proxies and load them in the context of refreshing the whole object. More information is provided in the implementation section.

Echo2 Framework

The Echo2 framework is a rich Web client framework based on the MVC pattern (more accurately the Model-Delegate variant of the MVC pattern) and is very similar to the standard Java Swing API. It has many basic components such as text inputs, different types of buttons, lists, as well as more advanced components like tables and tabsets. The EchoPointNG project extends some of the Echo2 components and adds new ones, such as a tree component.

A major problem with Echo2 is that creating new components not built as composites of the existing components is complicated requiring a mix of Java and Javascript. Working with the Echo2 Javascript can be difficult. For example, there was a bug in the interaction of the tree component and drag-and-drop used in Requel that took days to debug. Echo2 uses the Javascript “eval” command as a form of reflection to dynamically call extension code. The Firebug Javascript debugger could not identify the code across the boundary and display the full call stack.

Another problem with Echo2 is dealing with restrictions of what components can be nested in other components. For example an Echo2 panel can only contain a single component. It requires using specialized layout components like grids, rows and columns to contain multiple children. However, grids, rows and columns cannot contain panels. The EchoPointNG extended panel does support multiple children components so it is used as the basis for Requel’s panels.

Echo2 has its own XML based stylesheet language for styling the look of components. It is somewhat complicated to work with primarily because there is little documentation and the style properties are component specific. Also, only one stylesheet can be used at a time so all the styles for all panels must be contained in a single file. Despite these issues Requel is configured so that the styles of all the components are configurable using an Echo2 stylesheet.

Hibernate Validator

Validation of user input is important, but coding it by hand is tedious to implement and can make code fragile. The Hibernate Validator simplifies validation by allowing constraints to be defined as annotations on the properties of the persistent entities and performing the validation in the persistence engine when an entity is about to be saved.

Java API for XML Binding (JAXB)

JAXB is a framework for working with XML data files with the promise of not having to understand XML or XML processing APIs (Ort & Mehta 2003.) It uses a binding compiler to generate a set of Java classes representing the XML data from an XML Schema (XSD) file or a schema generator to create an XML Schema from a set of Java classes. It works for both input and output of XML data using an “Unmarshaller” to read the XML file and generate a graph of Java objects and a “Marshaller” to write out a graph of Java objects.

The initial JAXB 1.0 API was oriented towards the XML data and schema, and not the object model. It was really an API for wrapping an XML document in JavaBean objects and not a framework for importing and exporting a Java object model. Version 2.0 of the API added facilities for starting from the Java side. JAXB 2.0 adds a schema generator, a set of annotations for controlling the output of the XML, and the ability to add custom adapters that alter JAXB’s view of objects (Goncalves, 2007.)

The advantage of using annotations to control the mapping to XML with the ability to automatically generate a corresponding XSD file makes JAXB seem like an efficient and time saving approach.

JAXB has a few issues that made it a little hard to work with. The biggest short coming was with the Unmarshaller callback mechanism for doing pre and post processing of objects as they are loaded. JAXB supports a reflection-based method where it looks for before and after unmarshall methods on each object and a listener based method that puts the processing for all objects in a single class. The listener supports processing of objects just after they are created, but before they get filled in with data, and after they have been filled in and all child elements have been processed.

What Requel required was the ability to replace new object instances created during unmarshalling with existing persistent instances from the database, particularly for users and permission types embedded in the project XML files. Unfortunately the reflection-based method only supported methods that took the Unmarshaller and the parent object as parameters and there was no way to pass a repository through the Unmarshaller for an object to look up a corresponding persistent object. The listener method would support using a repository, but then the processing for all types of projects would need to be implemented in the listener or a set of additional classes for each of the domain object types. The solution ended up being a custom reflection-based method using a listener. The listener inspects each object for methods named "beforeUnmarshall" and "afterUnmarshall" with different parameter signatures and calls the methods it finds passing the appropriate objects.

Stanford Parser

The Stanford parser is actually a collection of parsers that generates both constituent and dependency parses. It has a built in sentence detector, tokenizer and tagger, although the sentence detector seems to treat most periods as ending a sentence. The Stanford parser uses probabilistic context free grammars that may or may not include lexical information (Klein &Manning, 2003.) The novelty of the Stanford parser is that is uses separate models for the constituent and dependency parsing and then combines them into a final lexicalized parse (Klein &Manning, 2003.)

The only problems with the Stanford parser are that it uses a lot of memory to hold the models and the time to parse grows quickly relative to the length of the sentence being parsed.

OpenNLP

OpenNLP is a collection of natural language processing tools implemented in Java including a sentence segmenter, tokenizer, tagger, constituent parser, and named entity recognizer. In the end only the sentence segmenter performed better than the corresponding Stanford tools and is the only component used.

The sentence detector uses a probabilistic machine learning method called maximum entropy to train a model from example sentences.

WordNet SQL builder

The WordNet SQL builder combines WordNet, VerbNet, and Extended WordNet data into a MySQL database.

WordNet (Fellbaum, 1998) is a lexical database containing over 140 thousand words, over 115 thousand concepts called synsets that give meaning to the words, and over 240 thousand semantic relationships between the concepts. The WordNet database is used for spell checking, word sense disambiguation, semantic role labeling, and vagueness checking.

VerbNet is a verb lexicon of a subset of the verb sense in WordNet that contains syntactic and semantic frames identifying the thematic roles of the words related to the verb in a sentence (Schuler 2005.) The frames are organized into classes of verbs that share similar uses and thematic roles. For example, below is the syntax part of one of the frames for the verb "accompany". It shows that the noun phrase preceding the verb has the "Agent" role and the noun phrase following the verb has the "Theme" role.

```
<NP value="Agent"></NP><VERB/><NP value="Theme"></NP>
```

Some frames contain selection restrictions on the types of nouns appropriate for that frame. Examples include organization, animate, location, or human. The frames are useful for semantic role labeling, but could also be used for word sense disambiguation to help select verb senses by the syntax of the sentence and category of the nouns filling the roles.

Extended WordNet takes the information in WordNet and processes it to generate the syntactic parse of each concepts definition, a logical representation of the meaning of the definition, and identify the sense of each word in the definition.

Implementation

Requel is a Web application based on the Java Enterprise Edition (JEE5) and Spring Framework platforms, intended to run in a Web container such as Tomcat. The Requel system is implemented using over 600 Java classes. This section describes the overall structure of the code and the implementation of some of the interesting aspects of the system.

Package Structure

Requel has three primary domain packages for the core application domain, natural language processing, and the user interface framework. Requel has two support packages for general command processing and repository support.

Package	**Content**
edu.harvard.fas.rregan.command	This packages contains the base interface for all commands, the base interface for command factories, the interface for the command handler, and the implementation of the default handler, a handler that does exception mapping from external exceptions to Requel specific exceptions, and a handler that catches database lock related exceptions and re-executes the command.
edu.harvard.fas.rregan.nlp	This package contains the classes for modeling the processed text, adapters for external tools, and the implementation for a lemmatizer, word sense disambiguator and semantic role labeler. A sub-package contains the interface to the lexical database (WordNet, VerbNet, etc.)
edu.harvard.fas.rregan.repository	This package contains a base repository implementation for JPA with functions for reloading an object from the database, making an object persistent, and deleting objects. This package also contains the implementation of the domain object proxy and AOP advice used to wrap persistent objects so they can refresh themselves from the database and load lazy collections.
edu.harvard.fas.rregan.requel	This package contains the domain logic for managing users, working with requirements and projects, and working with annotations. Each domain has its own sub-package. There is a package that contains the user interface components for all the domains.
edu.harvard.fas.rregan.uiframework	This package contains all the code for the custom user interface framework built on top of Echo2.

Table 24: Requel Source Code Packages

Application Initialization

The Requel system uses JEE ServletContextListeners to create the database, initialize the Spring Framework, and load data into the database when the application is loaded. Figure 12 shows a sequence diagram describing the startup process.

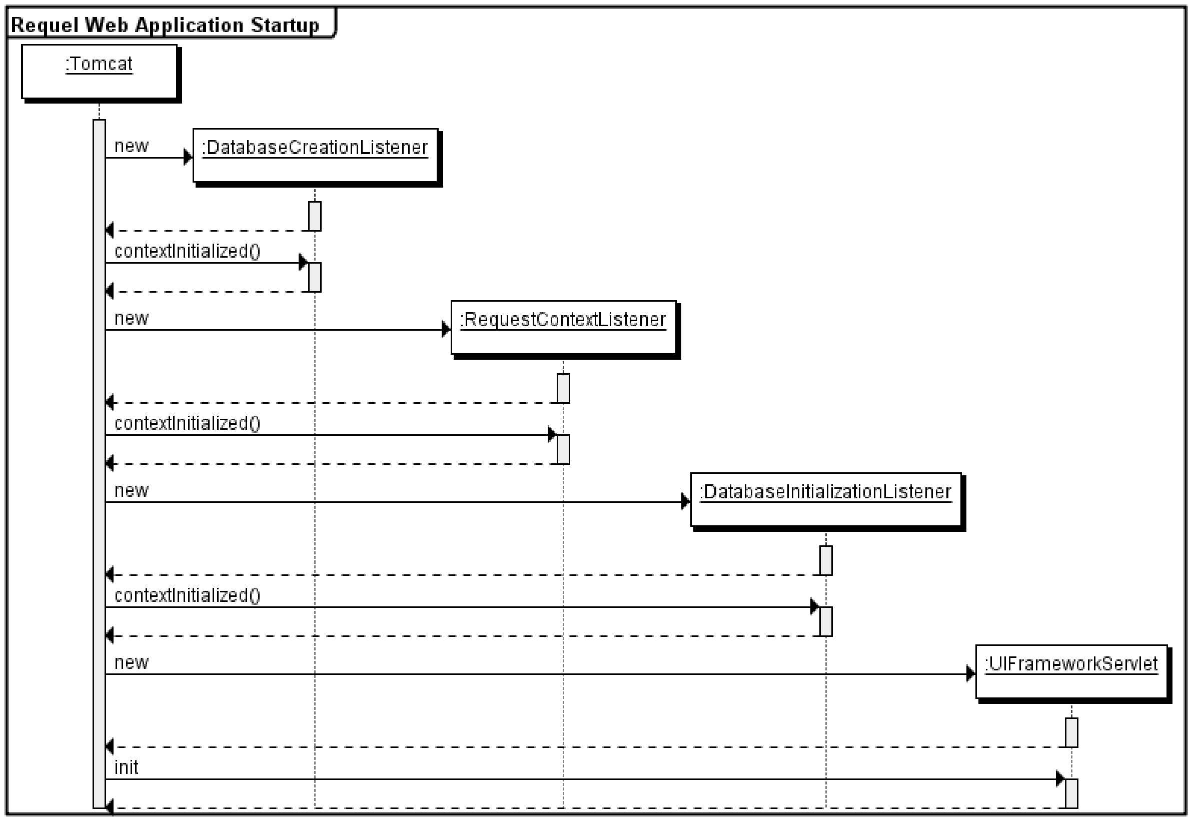


Figure 12: Application Initialization

The DatabaseCreationListener attempts to connect to the database specified for the system, and if the connection fails because the database doesn’t exist, it creates the database.

The RequestContextListener is part of the Spring Framework. It loads the Spring configuration files and scans the classpath for classes annotated with annotations. It initializes Hibernate and the Hibernate Schema Update automatically generates the database schema and creates the.

The DatabaseInitializationListener gets the DatabaseInitializer object from the Spring Framework context. The DatabaseInitializer is configured with a set of EntityInitializers for loading the dictionary data, creating the default users, and initializing the user and stakeholder permissions. Figure 13 shows the classes involved in the process.

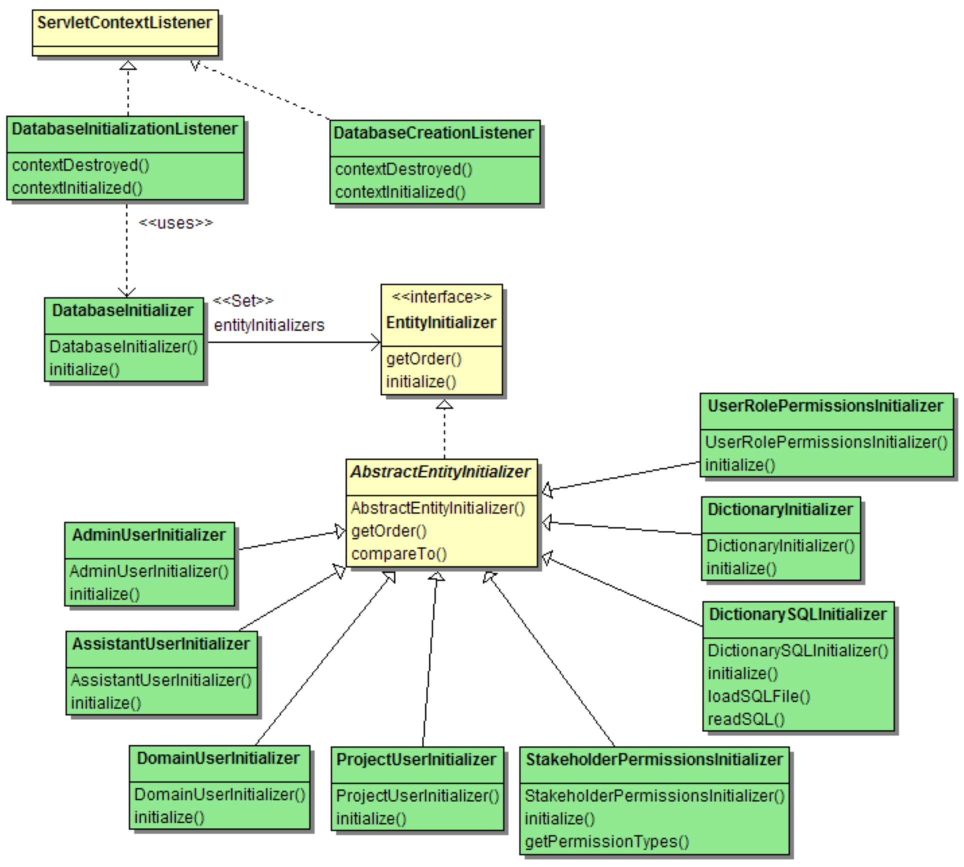


Figure 13: Database Initializers

User Interface Framework

A downside of the Echo2 framework is that it only provides low level components, which requires significant effort to create higher level constructs such as forms and navigation. Another issue, inherent in the observer pattern used for event passing, is that the controllers need to be explicitly wired together with the components they listen to. That is not such a big problem for components and controllers that are only concerned with activity internal to a panel or form, but when panels and forms need to interact, managing the relationships can add complexity and unwanted dependencies.

The UI Framework builds upon the Echo2 framework and provides higher level components including screens, panels and editors. The framework includes facilities for screen and panel management, entity and event type navigation, and event based message passing via a message broker. Figure 14 shows the core panel, screen and management classes of the framework. The key components are the Panel, PanelFactory, and PanelManager. A panel factory is used by a panel manager to create new instances of a panel when the manager receives an open event, or to display an existing panel if the open event references the panel directly.

The UI Framework uses the Spring Framework to wire together the event dispatcher, screens, panels and controllers when a user first access the system as shown in Figure 15. The use of the Spring Framework and the event dispatcher allows the panels to be completely independent of each other. For example, an edit button on the goal navigator panel fires an open event for editing a goal. The panel manager receives the message and searches for an existing panel already open for editing the goal, and if one isn’t opened it searches for a panel factory for creating a new panel for editing the goal. A new goal editor panel can be written and added to the application by simply changing the Spring Framework panel configuration to refer to the new panel. None of the existing panels need to change to use the new panel.

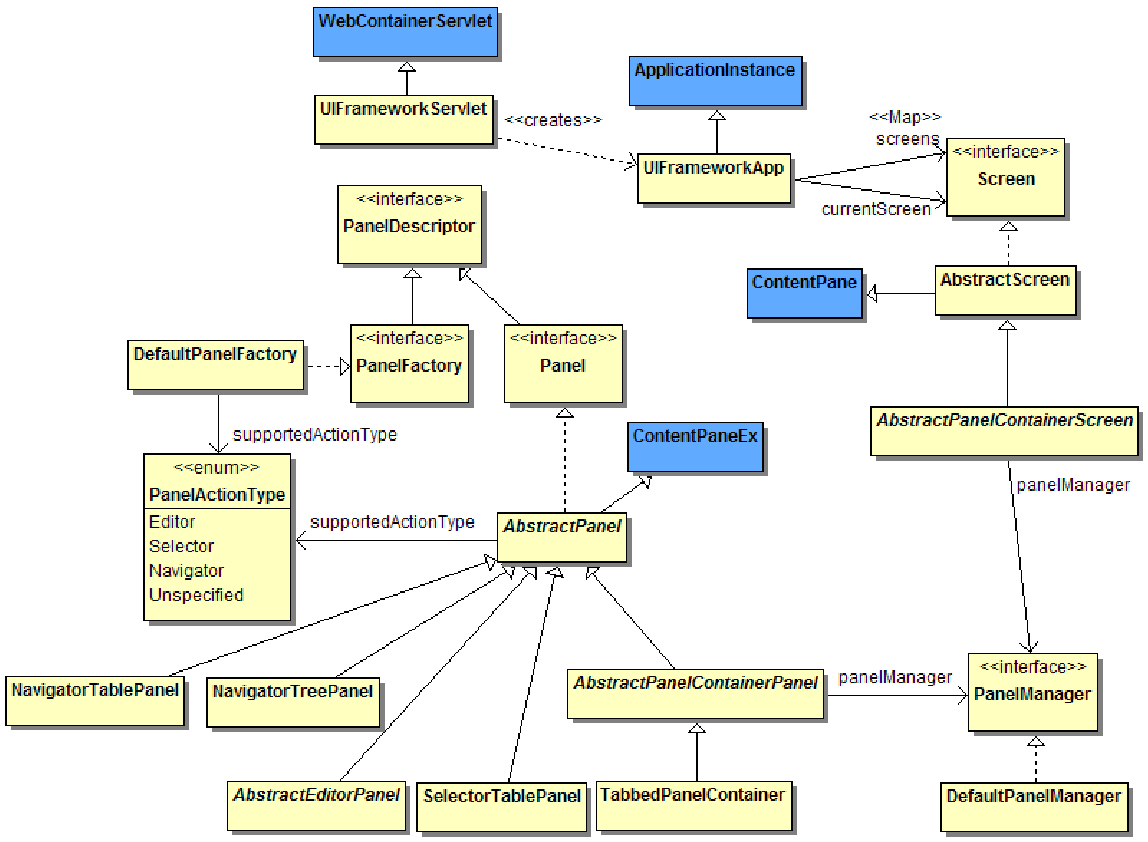


Figure 14: UI Framework Screens and Panels

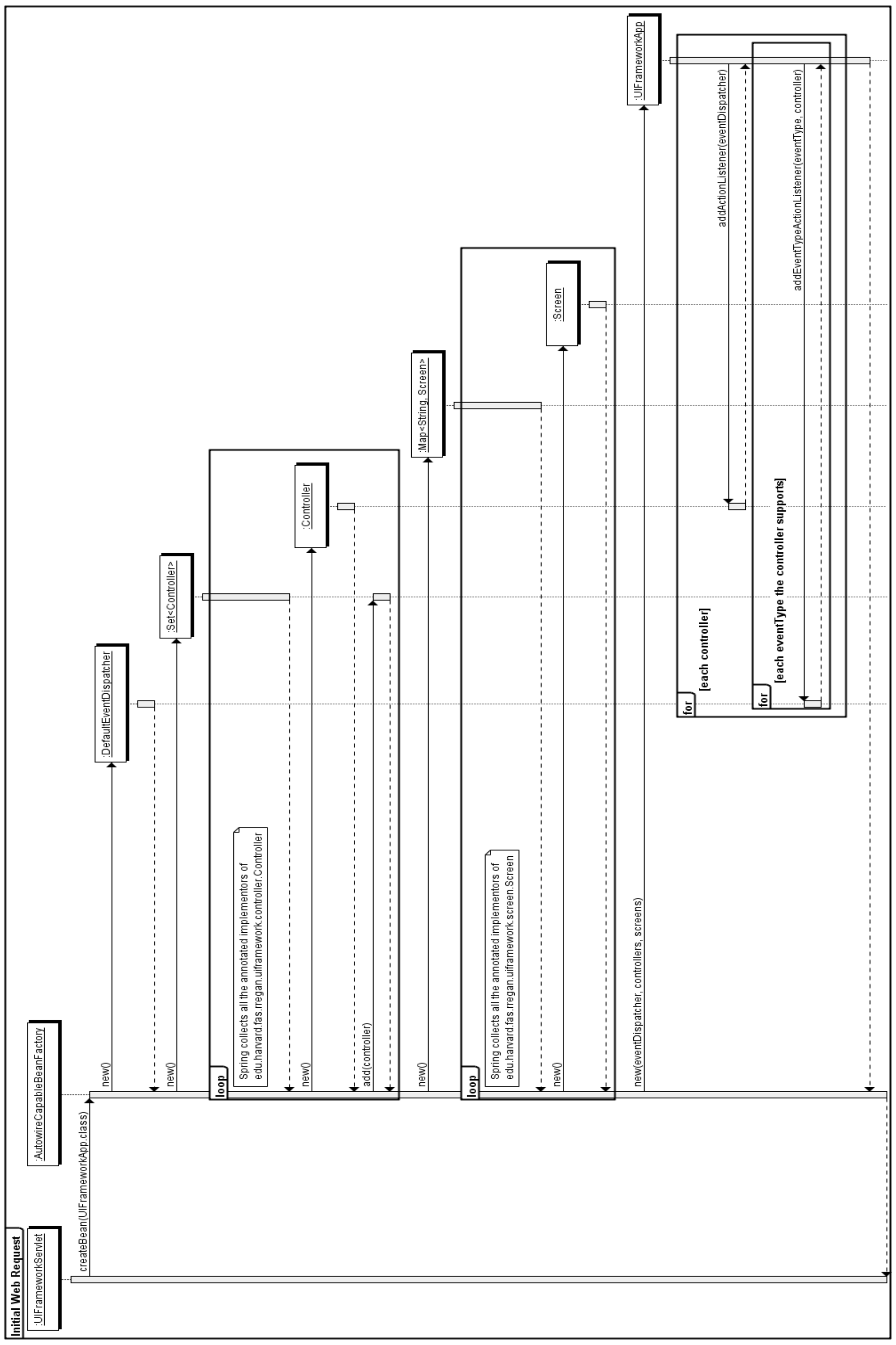

Initial Web Request
:UIFrameworkServlet
:AutowireCapableBeanFactory
createBean(UIFrameworkApp.class)
new()
:DefaultEventDispatcher
new()
:Set<Controller>
loop
Spring collects all the annotated implementors of edu.harvard.fas.rregan.uiframework.controller.Controller
new()
:Controller
add(controller)
new()
:Map<String, Screen>
loop
Spring collects all the annotated implementors of edu.harvard.fas.rregan.uiframework.screen.Screen
new()
:Screen
new(eventDispatcher, controllers, screens)
:UIFrameworkApp
for
[each controller]
addActionListener(eventDispatcher)
for
[each eventType the controller supports]
addEventTypeActionListener(eventType, controller)

Figure 15: UI Framework Initialization

The Echo2 Framework uses event based message passing to indicate simple user interface component actions such as clicking a button, selecting an item in a list, or changing the contents of a text box. The UI Framework builds on this mechanism with a richer set of event messages for navigational events such as login, logout, open a screen, open an edit panel, open a panel to search/select an entity of a particular type, or to indicate that an entity has been changed or selected. Figure 16 shows the classes that make up the events and dispatching mechanism.

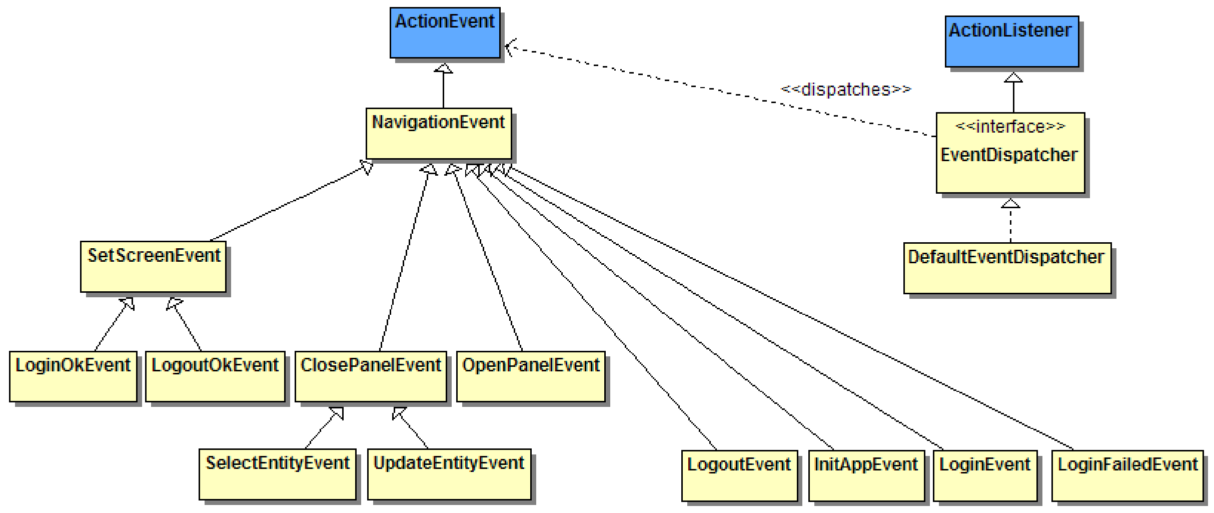


Figure 16: UI Framework Events

The framework uses both local and global events and listeners. The local events and controllers are for actions specific to a panel or composite UI component. For example a delete button on an editor is registered to a local controller on the panel. The listeners may be distinct classes such as the SubmitLoginListener or may be implemented directly by a panel. They are registered directly with the components that generate the events using methods such as addActionListener(). The listener may handle the event locally (within a single panel) by acting on the panel to change a data value on the panel or invoke a method on the panel, for example the save method on an editor panel. A local listener may generate a message like a Login Event and pass it through the Event Dispatcher to a global level controller. Global controllers listen for navigational events fired by the event dispatcher. A Panel Manager is an example of a global controller that listens for requests to open or close panels. Having two levels of messages improves the performance of the event dispatcher by not having it deal with the messages passed between components on a panel. Figure 17 and Figure 18 shows the sequence of event processing from the local level to the global level for the login process.

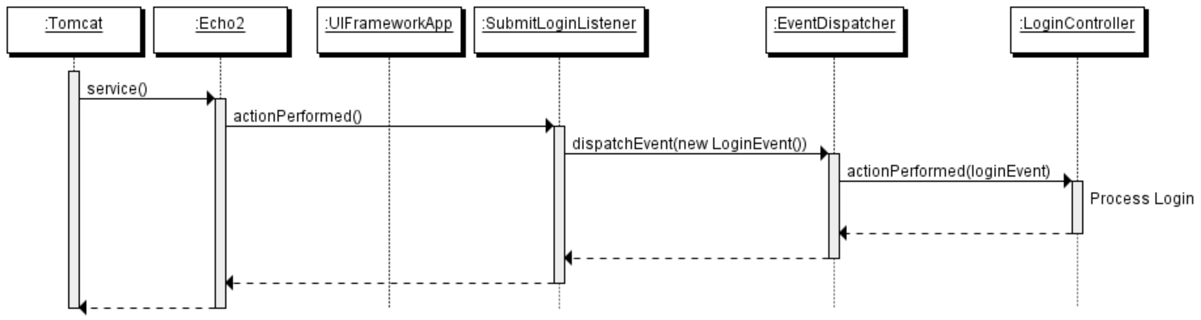


Figure 17: Event Processing Sequence of Login up to the EventDispatcher

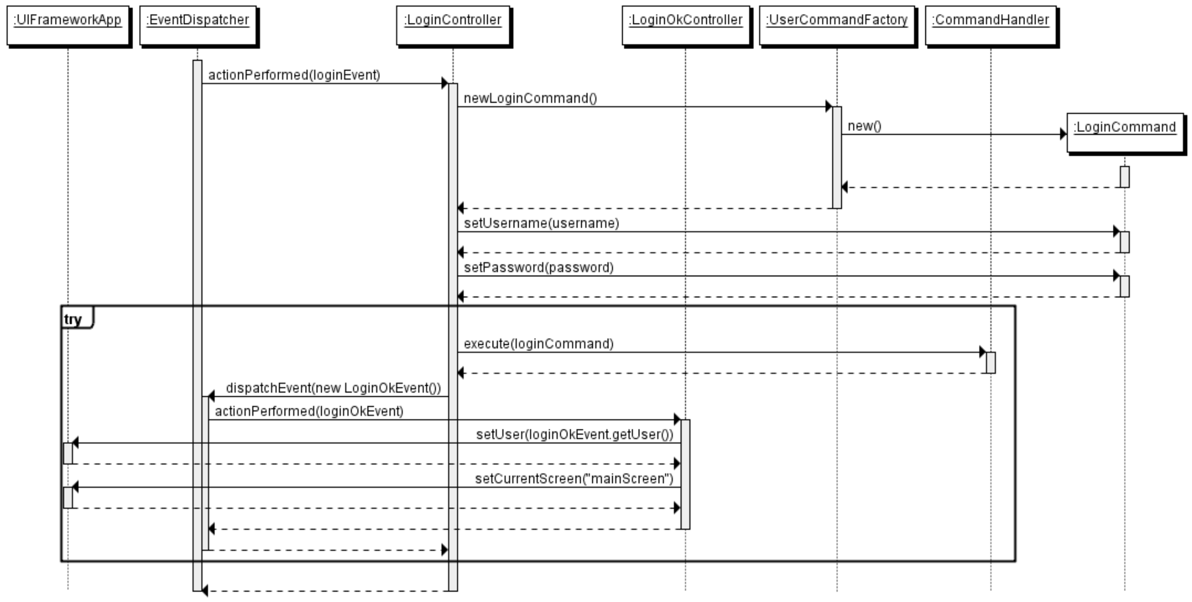


Figure 18: Event Processing Sequence of Login after the EventDispatcher

Requel User Interface

The user interface builds on top of the UI framework and Echo2 to create the panels for navigating around and editing the domain objects. Figure 19 shows the main screen of the Requel application.

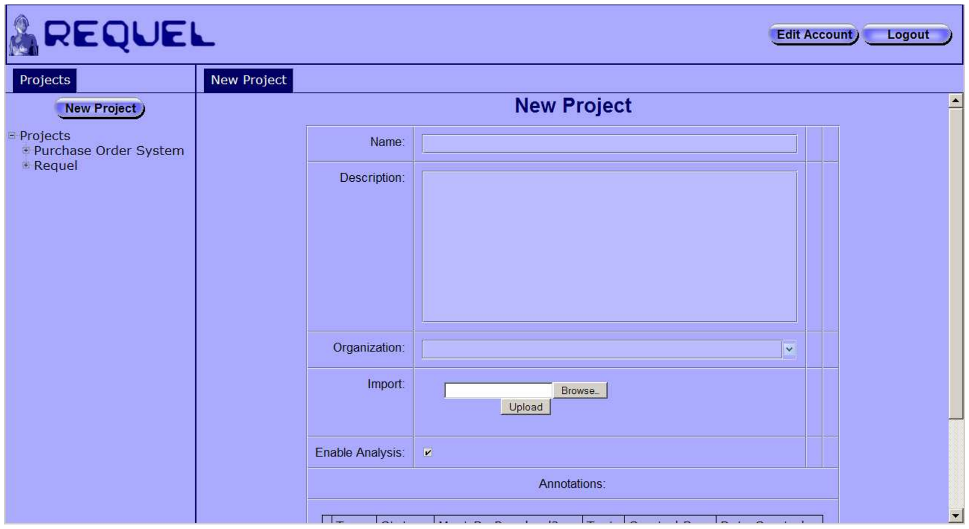
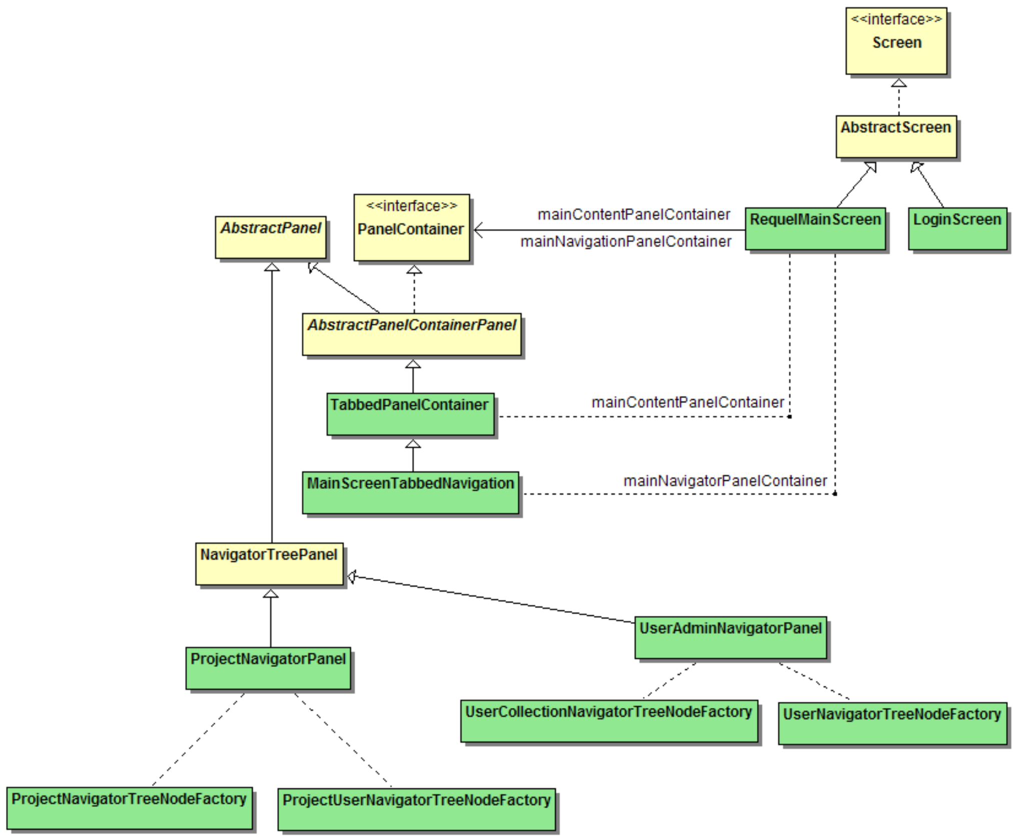


Figure 19: Requel Main Screen

Figure 20 shows the classes involved with the main screen and navigation. The tabbed navigation on the left side of the screen is a managed by a MainScreenTabbedNavigation panel container. The panel contains one or more ProjectNavigatorPanels depending on the roles of the user. The trees in the panels are configured using TreeNodeFactories specific to the type of target object being navigated. For example the Projects navigator panel uses two factories to build different parts of the tree. The main target of the panel is the ProjectUser role and the ProjectUserNavigatorTreeNodeFactory creates nodes for each project the user is assigned. The sub-nodes for each project are created by the ProjectNavigatorTreeNodeFactory.



<<interface>> Screen
AbstractScreen
RequelMainScreen
LoginScreen
<<interface>> PanelContainer
AbstractPanel
AbstractPanelContainerPanel
TabbedPanelContainer
MainScreenTabbedNavigation
NavigatorTreePanel
ProjectNavigatorPanel
UserAdminNavigatorPanel
UserCollectionNavigatorTreeNodeFactory
UserNavigatorTreeNodeFactory
ProjectNavigatorTreeNodeFactory
ProjectUserNavigatorTreeNodeFactory
mainContentPanelContainer
mainNavigationPanelContainer
mainContentPanelContainer
mainNavigatorPanelContainer

Figure 20: Main Requel Screen Components

The right side of the screen as shown in Figure 19 is the main content panel that uses a TabbedPanelContainer. This panel displays the domain object specific editors, selectors and navigator lists. The use of a TabbedPanelContainer allows multiple panels to be open at once so that users can change which panel is shown by clicking on the tab button at the top.

Command Handler

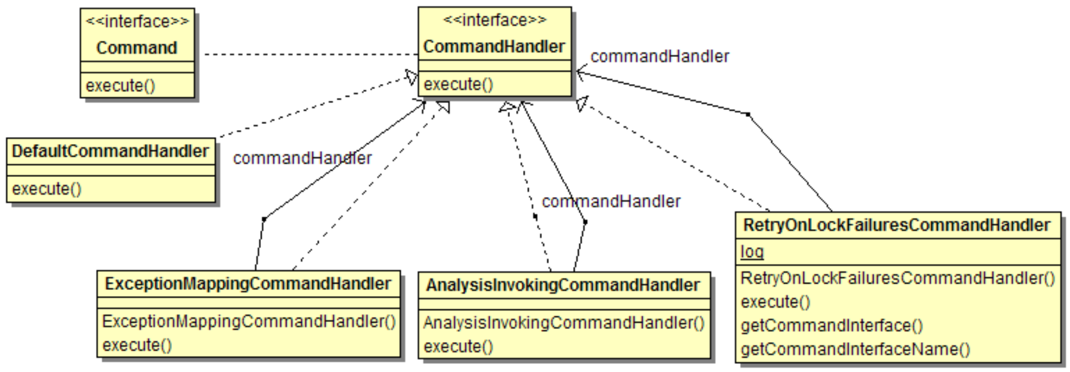
The command handler is the primary component in the Command Processor architectural pattern described in the architecture. The base handler is used to demark transaction boundaries; a transaction starts when the execute() method is invoked and ends when it returns. Requel uses decorators to alter the functionality of the default command handler.

One decorator catches exceptions thrown during the execution of a command and transforms them into Requel specific entity exceptions. This is done to simplify handling of exceptions generated by different packages. For example, exceptions may be thrown originating from JDBC, Hibernate, JPA, or the Spring Framework. Handling each type of exception throughout the application would overly complicate the code.

Another decorator is used to detect command execution failures due to database contention issues and retry the execution of the command until it doesn't encounter a database contention related failure or a preset maximum number of retries is tried.

The final decorator is specific to commands for editing requirements. It detects if the resulting domain object of a command should be analyzed and invokes a special method on the command to start the analysis after the commands execute method completes successfully and the transaction ends.

Figure 21 shows the class structure of the command handlers. Each of the decorator handlers has a reference to the inner factory.



<<interface>>
Command
execute()
<<interface>>
CommandHandler
execute()
commandHandler
DefaultCommandHandler
execute()
commandHandler
commandHandler
ExceptionMappingCommandHandler
ExceptionMappingCommandHandler()
execute()
AnalysisInvokingCommandHandler
AnalysisInvokingCommandHandler()
execute()
RetryOnLockFailuresCommandHandler
log
RetryOnLockFailuresCommandHandler()
execute()
getCommandInterface()
getCommandInterfaceName()

Figure 21: Command Handlers

Figure 22 shows the sequence of execution for an edit command from a panel through three handlers indicating the transaction part and exception handling.

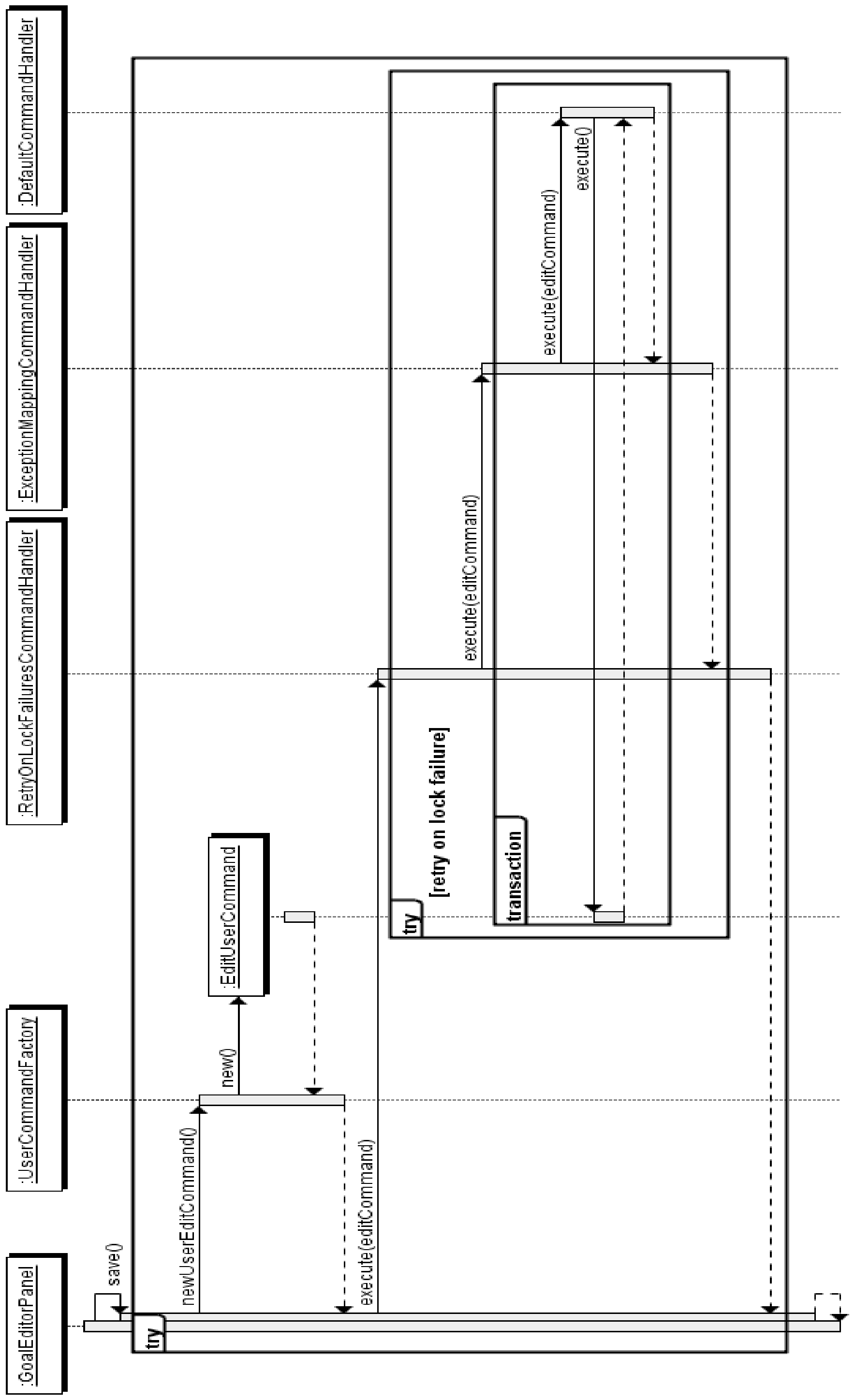


Figure 22: Command Handler Sequence

Command Factories

Spring manages the creation of most of the components used in the system except for commands. This is because the commands are single-use objects, where most of the other objects, such as UI components that last for a full user session with the Web server or the repository objects (data access objects) that are singletons, that last until the server is restarted.

The command factories use the Spring Framework to create the commands through the ApplicationContextCommandFactoryStrategy. Each time a factory method is called a new command object is created and returned. Figure 23 shows the class structure of the command factories with the UserCommandFactory as a concrete example.

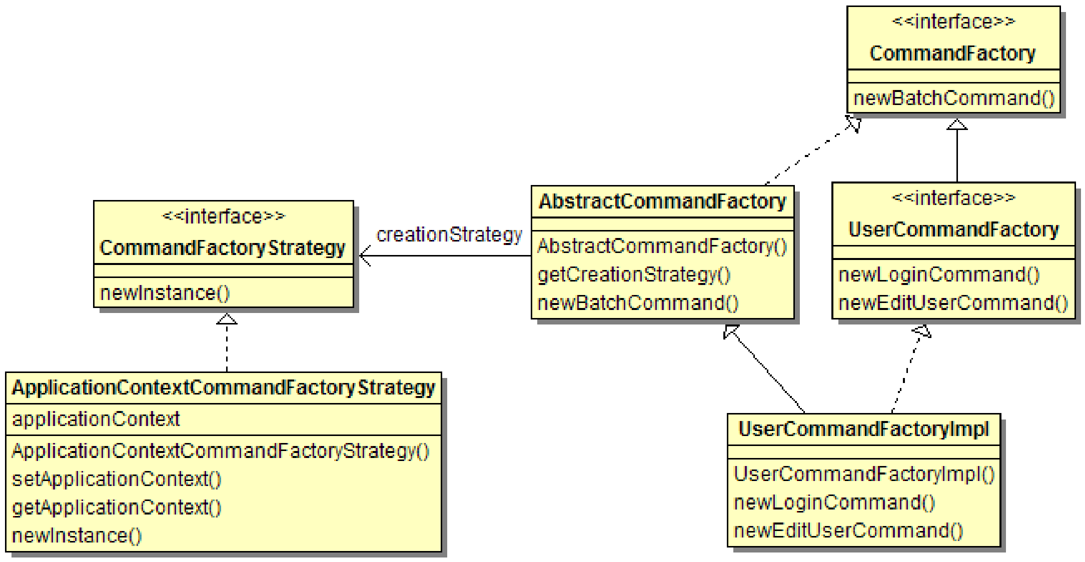


Figure 23: Command Factories

Domain Object Proxy

The domain objects of Requel, such as Users and Projects, are used as the model objects for most of the user interface framework panels. This ends up being a problem because the domain objects are persistent entities and when they get attached to the user interface components they lose their connection to Hibernate. This leads to two problems. First, the objects become out of date as they sit in the user interface because changes made by other users or the assistant and saved to the database don't get forwarded to the disconnected objects. Second, Hibernate uses proxies to stand in for other associated entities so that they don't have to be loaded unless needed. When the connection to Hibernate is lost the proxies lose their ability to load the underlying entities.

The solution to the problem is to add a proxy around each persistent object that keeps a copy of the persistent entity, periodically refreshing it to keep it fresh. When a method is called on the proxy to access a lazy loaded property, the call on the cached object is wrapped to catch lazy loading exceptions and retry loading in a transaction.

There were a lot of technical challenges in implementing the proxies. The largest challenge was how to get the domain objects wrapped in the proxies to begin with. Explicitly wrapping each object as it gets passed to the user interface layer is not practical. Custom entity loaders could have been added to Hibernate to wrap each object as it is returned, but always getting a proxy back is undesirable, especially in commands that are bound by a transaction. The solution was to use Spring's Aspect Oriented Programming (AOP) features to intercept objects as they were returned by a repository or command object and only if that call wasn't called from inside command in a transaction.

Figure 24 shows how the advice works in a command and returning an object from a command to a panel. Spring creates a proxy for the Command and Repository objects that applies the AOPInterceptor and DomaindObjectAdvice on calls to the methods. In the top half of the sequence the command calls a get() method on a repository to get a domain object. The DomainObjectAdvice detects that the call is being made from inside a command execute() method so it does not wrap the domain object. In the bottom half of the sequence the panel calls a getResult() method on the command to return the domain object. This time the DomainObjectAdvice doesn't detect the call is being made from inside a command execute() method so it creates a new DomainObjectWrapper and EntityProxyInterceptor (the proxy) and returns the proxy to the panel.

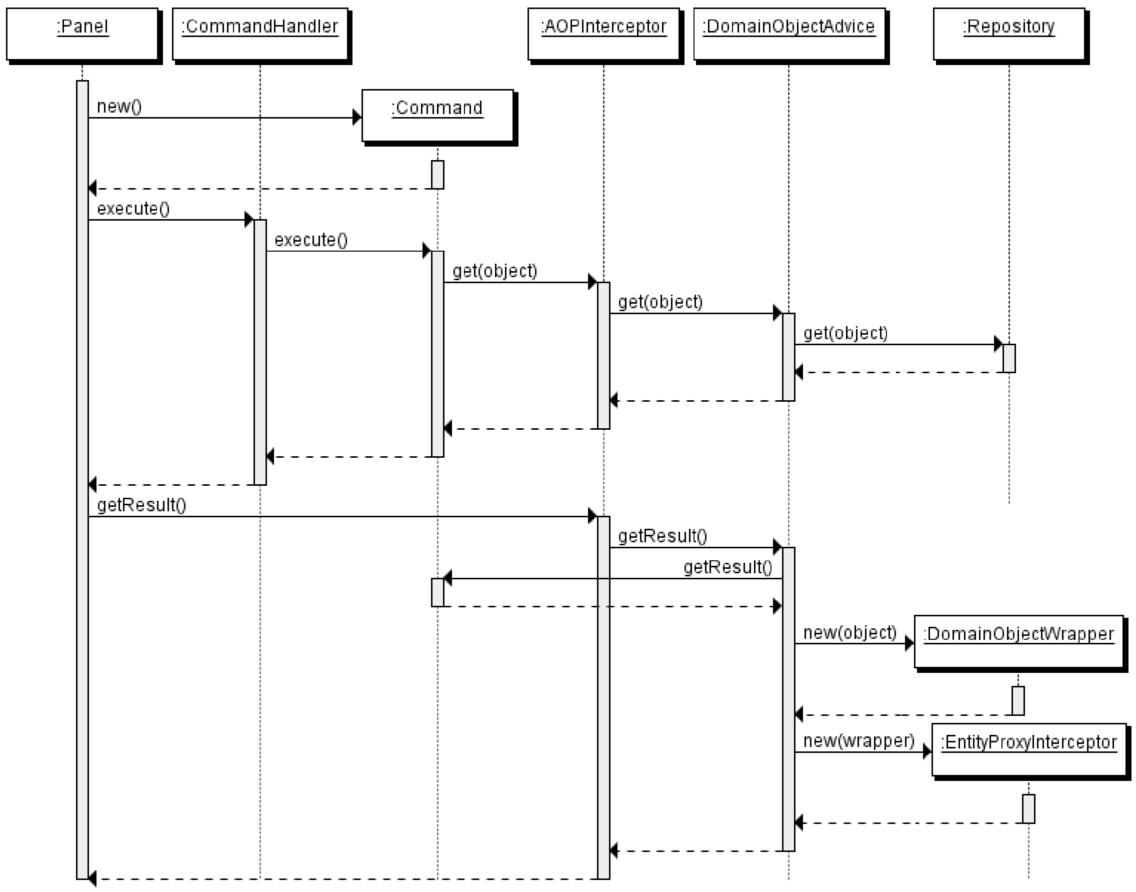


Figure 24: Domain Object Wrapping Advice in Command

Figure 25 shows the sequence where the panel calls the repository to load an object. Spring creates a proxy for the Repository objects that applies the AOPInterceptor and DomaindObjectAdvice on calls to its methods. The DomainObjectAdvice doesn't detect the call is being made from inside a command execute() method so it creates a new DomainObjectWrapper and EntityProxyInterceptor (the proxy) and returns the proxy to the panel.

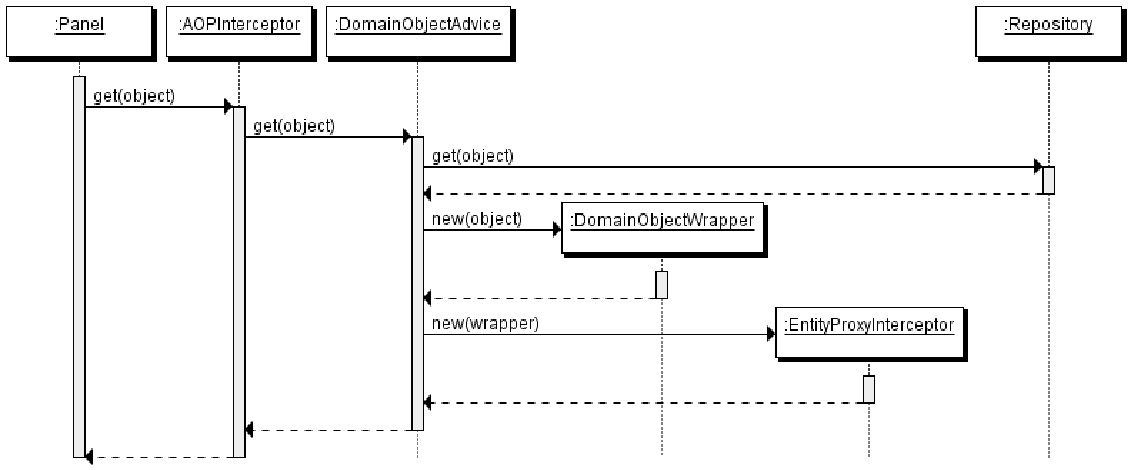


Figure 25: Domain Object Wrapping Advice out of Command

Word Sense Disambiguation

This is probably the toughest problem in NLP. Requel uses the word similarity and collocation disambiguation techniques described in the Natural Language Processing section of Chapter 1 as well as a relatedness method loosely based on the Lesk method. The relatedness method calculates the word similarity of the words in the definitions of the words being disambiguated. The intent of the last method was to get around the caveat that word similarity works only for words in the same part of speech. By using a level of indirection nouns and verbs are compared based on the words that define them.

The word similarity uses the WordNet information content formula defined by Seco, Veale and Hayes (2004) shown in Equation 1 using the WordNet hypernym-hyponym relationship. In selected test cases the word similarity measure worked quite well on nouns.

The collocation method searches for word sense pair combinations from semantically tagged WordNet definitions and SemCor sentences. Senses that were found to be co-located were given a higher ranking than those without a collocation.

The word sense disambiguator performs poorly. The collocation method did not appear to improve the quality of the disambiguation at all, but had a huge performance penalty so it was disabled in the final version.

Semantic Role Labeling

Requel uses VerbNet frames to match a verbs usage and then assign the roles based on the phrase structure to role mappings in the “syntax” of the frame. If a sentence doesn’t match a VerbNet frame Requel uses a simpler assignment based on the subject, direct object, indirect object, and preposition objects in dependencies identified by the parser.

In selected test cases of simply structured sentences the method worked fairly well, but due to the poor performance of the word sense disambiguator and more complex sentence structures in the test requirements the roles were mostly assigned by the simpler fall back method using the dependencies.

Chapter 4 User Guide

This chapter describes how to create and discuss requirements using the Requel system. The Requel system should already be installed on a Web server and your administrator should have given you a URL to access it. The Getting Started section defines Web browser requirements, how to connect to the system, and how to login.

Getting Started

To get started you need a Web browser, the URL to the Requel system, and a username and password. Mozilla Firefox version 3.0 or later is the only Web browser with which the system is certified to work correctly. To connect to the system, enter the URL of the application in the address bar of the browser.

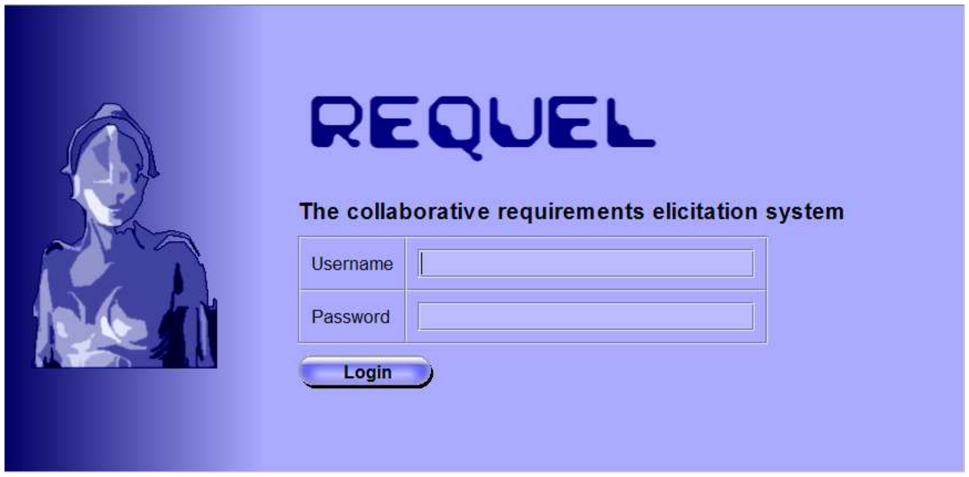


Figure 26: Login Screen

The system returns the login screen as shown in Figure 26. Enter the username and password supplied by your administrator. If your account is setup correctly you will see the main screen as shown in Figure 27.

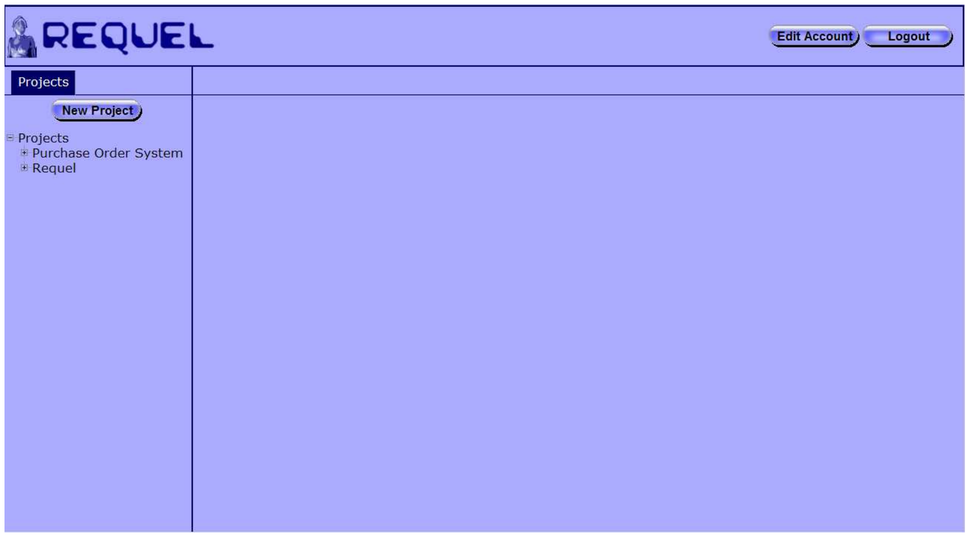


Figure 27: Main Screen

On the left hand side of the screen should be a tab labeled "Projects" with a tree containing the projects that you are assigned. If you have project creation authorization you will also see a "New Project" button above the projects. Figure 28 shows a close up of the project navigation tab.

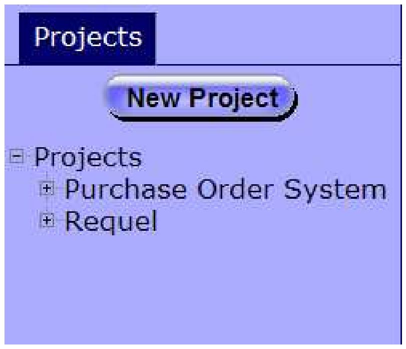
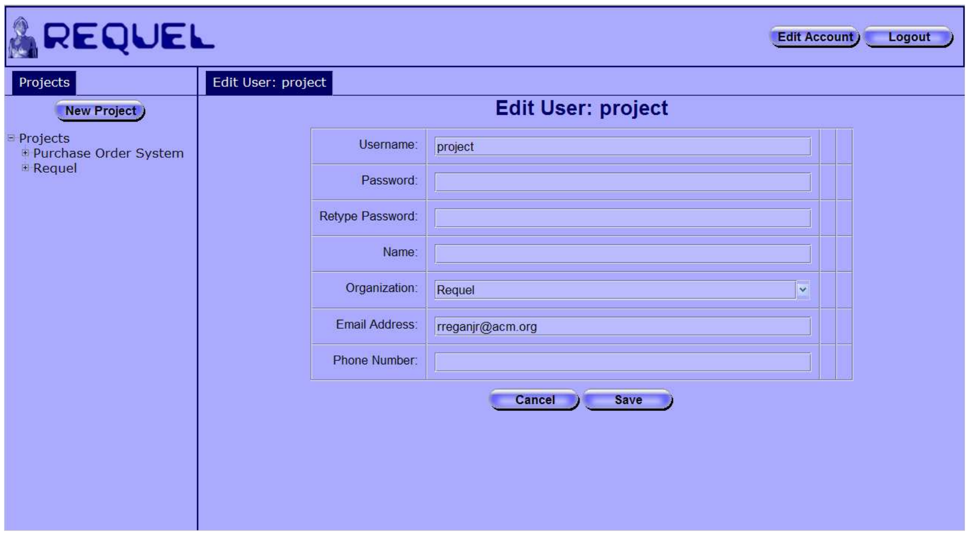


Figure 28: Projects Navigation Details

On the upper right hand of the screen are a “Logout” button and “Edit Account” button. The logout button clears out your user session on the server and returns you to the login screen.

The edit account button opens the “Edit User” screen for your system account as shown in Figure 29. When you first login you should change the password assigned by the administrator to something more secure and fill in any missing personal data, such as your name, email address, and phone number. Unless you have system administrator privileges you will not be able to change your username.



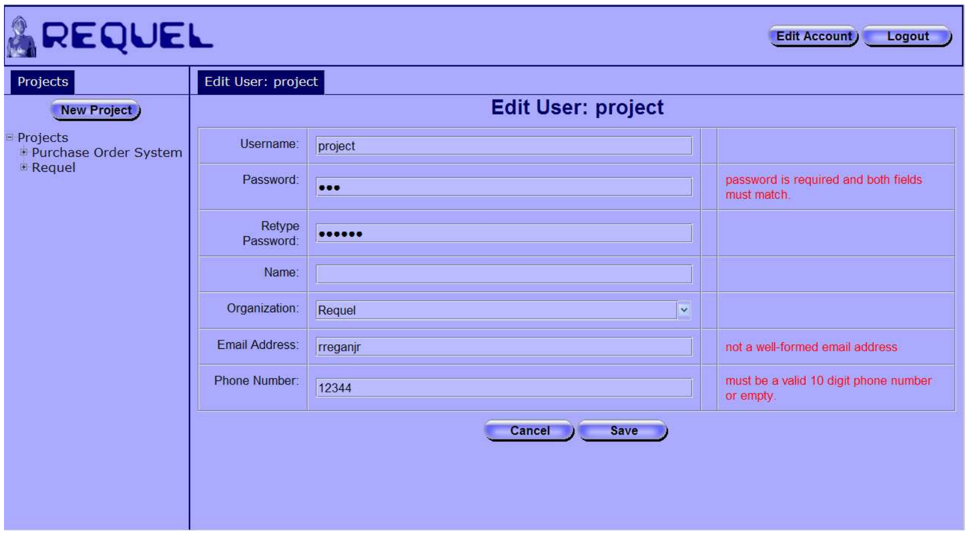
Edit User: project

Username:	project
Password:	
Retype Password:	
Name:	
Organization:	Requel
Email Address:	rreganjr@acm.org
Phone Number:	

Figure 29: Edit User Screen

After making changes to your account you must click the “Save” button at the bottom to submit the changes to the system. If everything is valid the edit user panel will close and your changes will be saved. If there are any problems with your data, such as the password and retype password fields don’t match, your email address is not formatted properly, or your phone number is not formatted properly, the system will display error messages next to the fields that need to be corrected as shown in Figure 30

31.



Edit User: project

Username:	project	
Password:	•••	password is required and both fields must match.
Retype Password:	••••••	
Name:		
Organization:	Requel	
Email Address:	rreganjr	not a well-formed email address
Phone Number:	12344	must be a valid 10 digit phone number or empty.

Figure 30: Edit User Screen with Errors

Working with Requirements

This section shows you how to create a project and add requirements to it. Requel doesn't impose any particular process for creating requirements, which makes it flexible, but the flexibility may make it harder for users to get started if they don't have a process for gathering requirements or an idea of how the requirements should be specified. To start you will need to create a project and the stakeholders that will create the requirements. For a simple project you may just want to use stories to describe the behavior of the system.

Creating a Project

Requel organizes requirements into projects. If you have project creation authorization the "New Project" button appears at the top of the project navigation tab as shown in Figure 28. Clicking the button opens the "New Project" screen show in Figure 31.

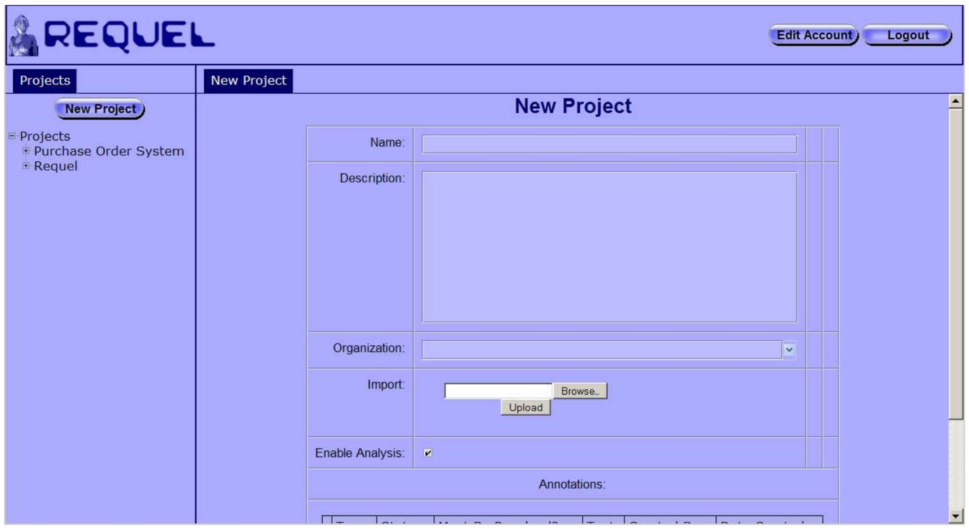


Figure 31: New Project Screen

The new project screen supports creating a new project from scratch or importing an existing project from an XML file. To create a new project from scratch enter a name, organization, optional description, and click the save button. The combination of project name and organization are required to be unique. The system verifies the values are valid, creates the project, and adds it to the project navigation tab. If there are validation problems the system puts a message to the right of the affected fields.

To import a project from an XML file, click the “Browse” button in the “Import” field and use the file upload dialog to navigate to the file to import as shown in Figure 32. Requel requires that a project name plus organization name is unique, so if you are importing a project that already exists in Requel, you must change the name by entering a new name in the name field. Only the name can be changed when importing a project; values supplied in the description and organization fields are ignored.

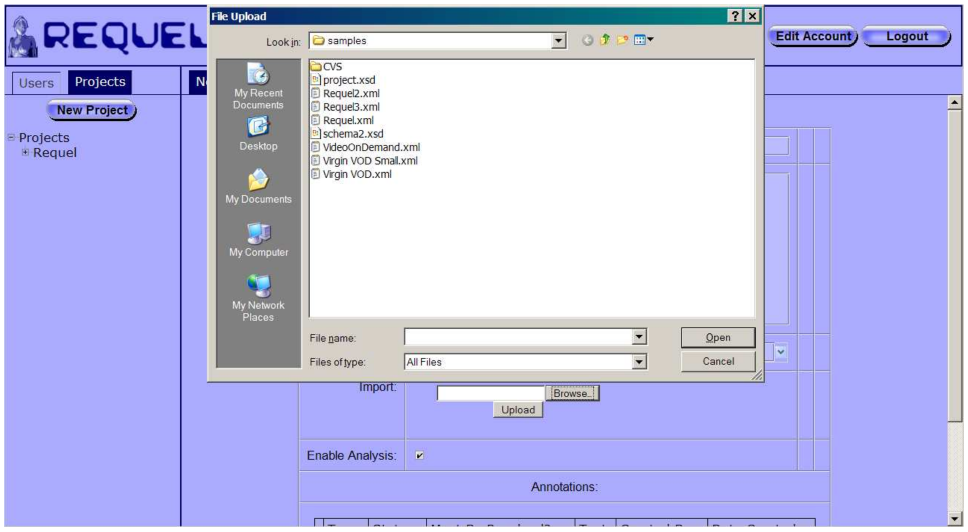


Figure 32: Project Import File Selection

When importing you have the option to turn off analysis of the project during the import using the "Enable Analysis" checkbox. This is useful for projects that were exported from the system and the elements have already been analyzed. Projects created by hand or by another tool in the project XML format should have analysis enabled.

After selecting the file click the "Upload" button to upload the project xml file to the server. Then click the "Import" button at the bottom of the screen. The system creates a new project and adds it to the project navigation tab.

Project Navigation

Figure 33 shows a close up of the project navigation tab with the "Purchase Order System" project node in a closed state and the "Requel" project node in an open state. Clicking on the project name opens the "Project Overview" panel for editing the name, description and organization of the project, as well as adding issues and notes to the project. Clicking on the + next to a project name in the project navigation tab expands the project node to display the links to the requirements elements.

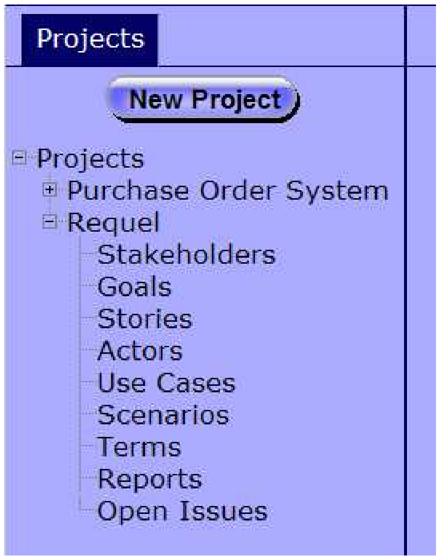
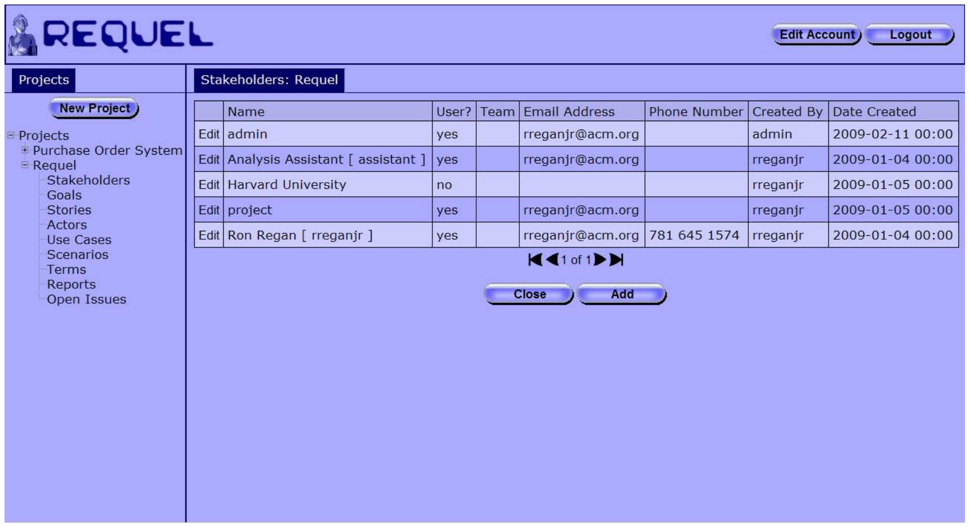


Figure 33: Project Navigation Tab

Creating and Editing Stakeholders

The first task to do after creating a new project is adding stakeholders for the users that will work on the project. Clicking on the "Stakeholders" link under a project in the navigation tab opens the stakeholder navigator that lists all the stakeholders on the project ordered by username as shown in Figure 34.

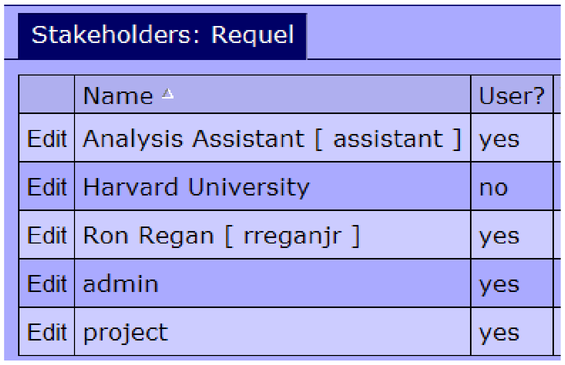


Stakeholders: Requel

	Name	User?	Team	Email Address	Phone Number	Created By	Date Created
Edit	admin	yes		rreganjr@acm.org		admin	2009-02-11 00:00
Edit	Analysis Assistant [assistant]	yes		rreganjr@acm.org		rreganjr	2009-01-04 00:00
Edit	Harvard University	no				rreganjr	2009-01-05 00:00
Edit	project	yes		rreganjr@acm.org		rreganjr	2009-01-05 00:00
Edit	Ron Regan [rreganjr]	yes		rreganjr@acm.org	781 645 1574	rreganjr	2009-01-04 00:00

Figure 34: Stakeholder Navigator

Users authorized to edit stakeholders will see an “Edit” link in the first column of each stakeholder entry in the table and an “Add” button below the table. If you are not authorized to edit stakeholders the first column contains a “View” link that opens the stakeholder editor, but with read-only access. Clicking on the column titles sorts the stakeholders by that column.



Stakeholders: Requel

	Name ▵	User?
Edit	Analysis Assistant [assistant]	yes
Edit	Harvard University	no
Edit	Ron Regan [rreganjr]	yes
Edit	admin	yes
Edit	project	yes

Figure 35: Sorting Stakeholders

The first click sorts them in ascending order as shown in Figure 35. The column indicates the sort order with an arrow next to the column name. Clicking again sorts the column in descending order. Note stakeholders without a name (admin and project) sort at the end.

There will always be at least two stakeholders, the person that created the project and the analysis assistant that the system uses to indicate issues and notes created by the system.

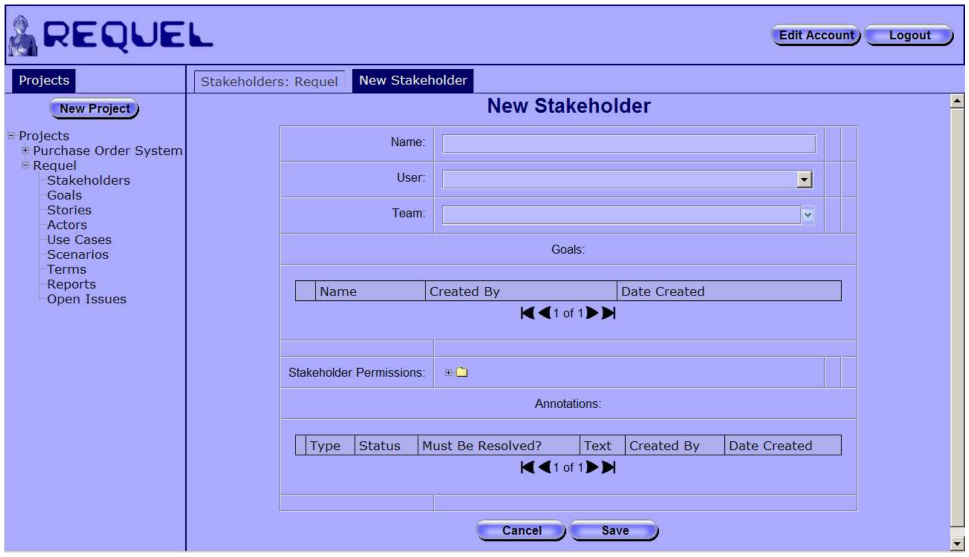


Figure 36: Edit Stakeholder Screen

Clicking the "Add" button on the stakeholder navigator screen opens the "New Stakeholder" screen shown in Figure 36. This screen is used for creating both user and non-user stakeholders for the project. To create a user stakeholder, leave the "Non-User Name" field empty and select a user from the "User" field drop down list by clicking the button at the right end of the list and clicking on a user name as shown in Figure 37.

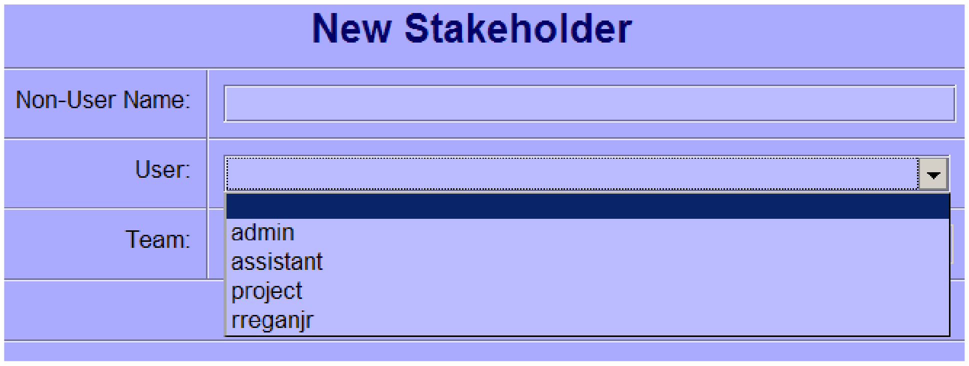
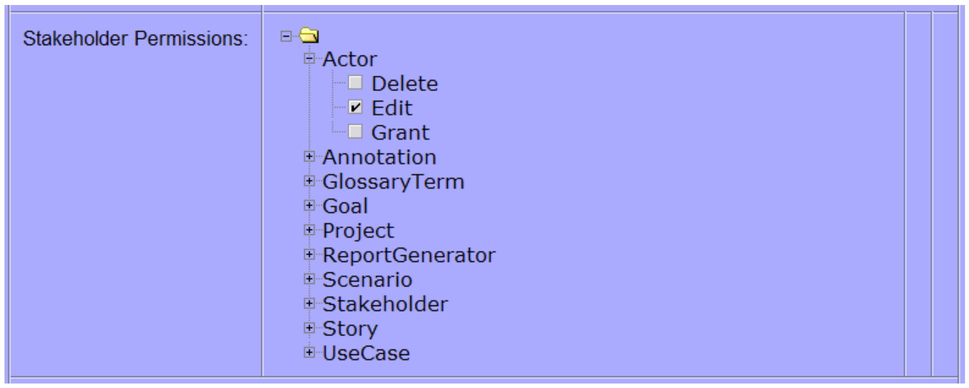


Figure 37: Selecting a User for a Stakeholder

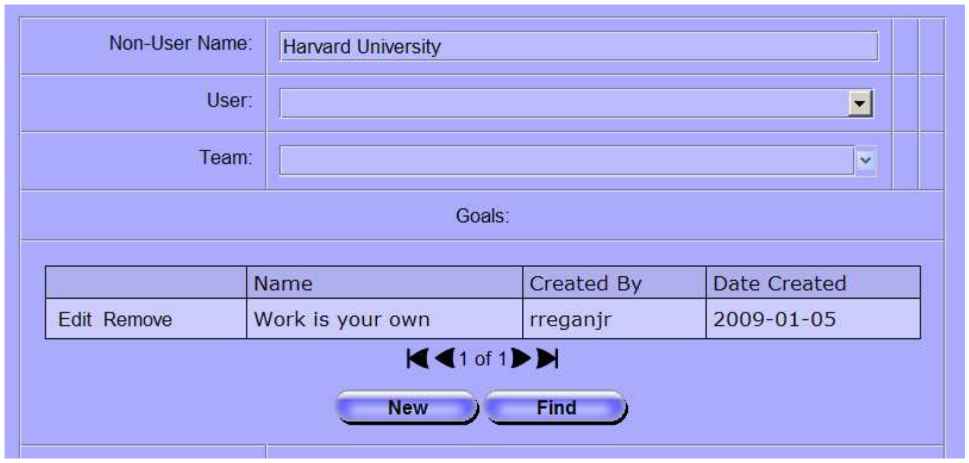
Only administrators can create users, if a person that you want to add to a project doesn't appear in the list they may not have a user account, you must ask the administrator to add them to the system with the "Project User" role. You may optionally enter or select a team name in the "Team" field. The team is for documentation purposes only. Customer and Supplier are examples of team names.


Stakeholder Permissions:

- Actor
  - [ ] Delete
  - [x] Edit
  - [ ] Grant
- Annotation
- GlossaryTerm
- Goal
- Project
- ReportGenerator
- Scenario
- Stakeholder
- Story
- UseCase

Figure 38: Stakeholder Permissions

User stakeholders can view all requirements elements on a project, but require explicit authorization to edit or delete requirements. The "Stakeholder Permissions" field contains a tree with nodes for each of the requirement element types. Each of those nodes having child nodes with permissions on actions such as "Delete", "Edit", or "Grant" and a checkbox as shown in Figure 38. Checking a box grants permission to the user to perform that action on that element type. For example in Figure 38 the checkbox for edit under actor is checked indicating the user can edit and create actors. The "Grant" action permission allows a user with stakeholder "Edit" permission to grant permissions for that element type to other users. For example, if stakeholder "A" has stakeholder edit permission and story grant permission, he or she can edit stakeholder "B" and grant or revoke edit, grant, and delete permission for actors to stakeholder "B". If stakeholder "A" does not have grant permission for story elements, he or she can not grant or revoke permissions related to stories to stakeholder "B."



Non-User Name:	Harvard University
User:	
Team:	

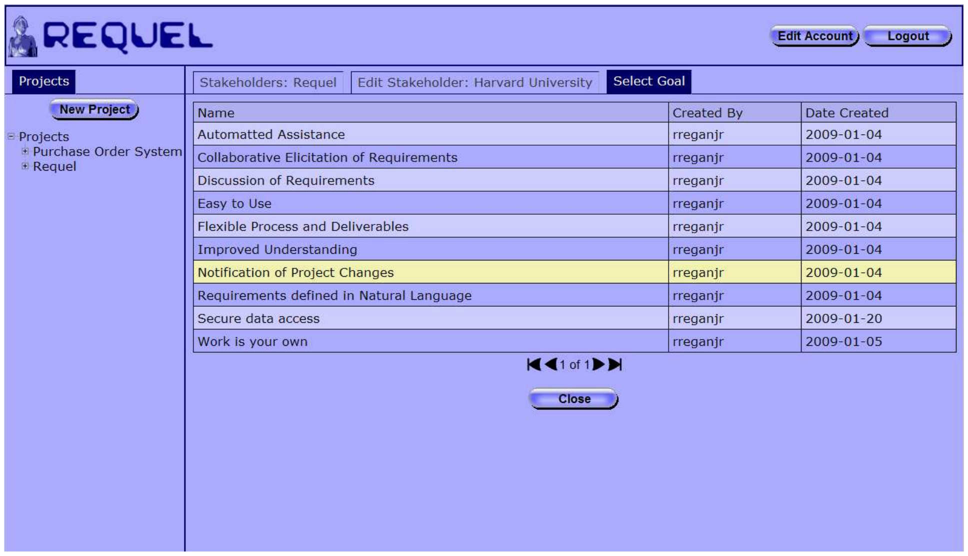
Goals:

	Name	Created By	Date Created
Edit Remove	Work is your own	rreganjr	2009-01-05

Figure 39: Stakeholder Goals

Non-user stakeholders represent organizations or regulatory bodies that don't have a user representative, but have interests or goals that the system must meet. Figure 39 shows a non-user stakeholder with one goal "Work is your own." Non-user stakeholders are created using the same screen as user stakeholders, but instead of selecting a user, a name is entered in the "Non-User Name" field. Non-user stakeholders may have a team. Stakeholder permissions don't make sense for non-user stakeholders.

After creating a stakeholder goals can be added. Both user and non-user stakeholders can have goals. A new goal can be added by clicking the "New" button, which opens the "New Goal" screen, or an existing goal can be added by clicking the "Find" button and selecting a goal from the goal selector panel. Figure 40 shows the goal selector panel with the selected goal highlighted.



Name	Created By	Date Created
Automatted Assistance	rreganjr	2009-01-04
Collaborative Elicitation of Requirements	rreganjr	2009-01-04
Discussion of Requirements	rreganjr	2009-01-04
Easy to Use	rreganjr	2009-01-04
Flexible Process and Deliverables	rreganjr	2009-01-04
Improved Understanding	rreganjr	2009-01-04
Notification of Project Changes	rreganjr	2009-01-04
Requirements defined in Natural Language	rreganjr	2009-01-04
Secure data access	rreganjr	2009-01-20
Work is your own	rreganjr	2009-01-05

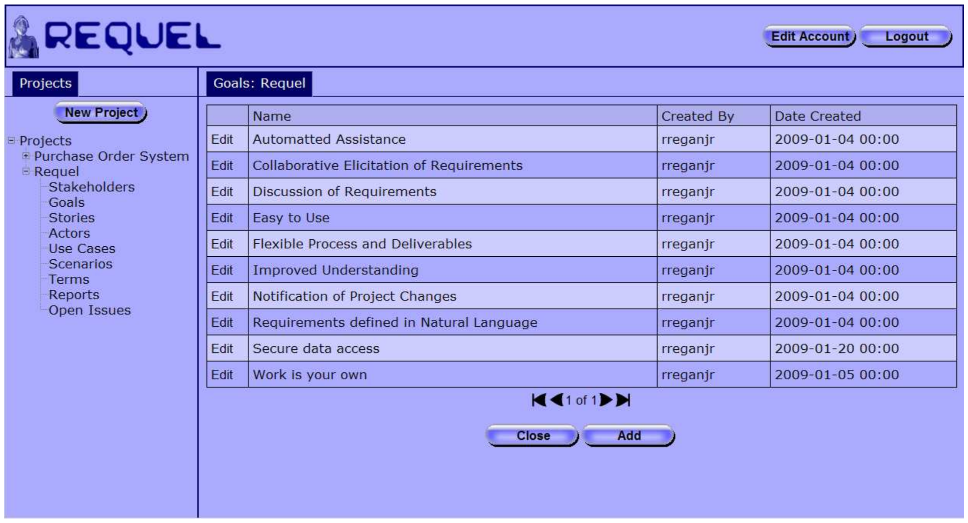
Figure 40: Goal Selector

The full goal can be viewed or edited by clicking on the "Edit" link in the first column of the goal table or removed from the stakeholder by clicking the "Remove" link as shown in Figure 39. When a goal is removed from a stakeholder it is not deleted from the system.

Creating and Editing Goals

Goals are used to indicate desired properties, qualities, constraints, features and functions that the system must support. For example, "Easy to Use" is a high-level quality goal; "Must run on Linux" is a technology constraint; "Support an IBIS style of discussion" is a feature.

Clicking on the "Goals" link in the project navigation tab opens the goal navigator screen shown in Figure 41. The goal navigator has a table with the name of each goal, who created it, and when it was created. If you are authorized to edit goals the first column contains an "Edit" link and below the table is an "Add" button. If you are not authorized to edit goals the first column contains a “View” link that opens the goal editor for read-only viewing.



	Name	Created By	Date Created
Edit	Automatted Assistance	rreganjr	2009-01-04 00:00
Edit	Collaborative Elicitation of Requirements	rreganjr	2009-01-04 00:00
Edit	Discussion of Requirements	rreganjr	2009-01-04 00:00
Edit	Easy to Use	rreganjr	2009-01-04 00:00
Edit	Flexible Process and Deliverables	rreganjr	2009-01-04 00:00
Edit	Improved Understanding	rreganjr	2009-01-04 00:00
Edit	Notification of Project Changes	rreganjr	2009-01-04 00:00
Edit	Requirements defined in Natural Language	rreganjr	2009-01-04 00:00
Edit	Secure data access	rreganjr	2009-01-20 00:00
Edit	Work is your own	rreganjr	2009-01-05 00:00

Figure 41: Goal Navigator Screen

Clicking the add button opens the “New Goal” screen shown in Figure 42. Goals have a name, which is a short one line description, and a text body that is the full description of the goal. The goal name must be unique in a project. After entering a name and text body, click the save button at the bottom of the screen to create the goal. If the name is unique the system creates the goal and adds it to the goal navigator table.

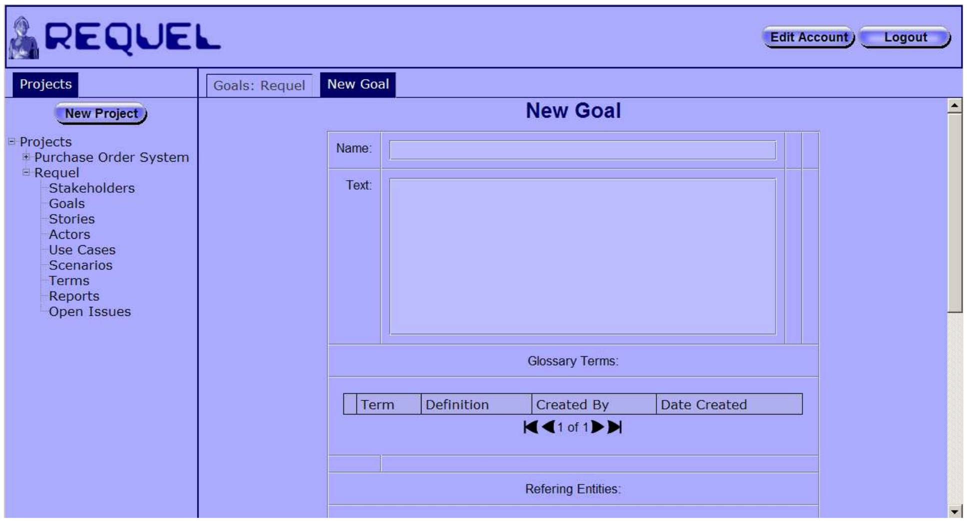
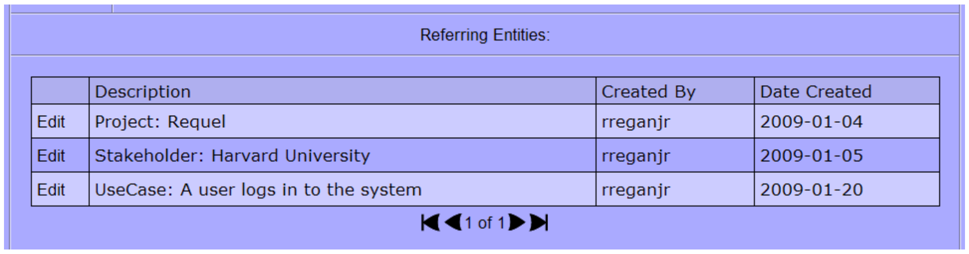


Figure 42: New Goal Screen

If the goal was created from another requirements element such as a stakeholder or use case, then the goal is also added to the goals of that element. All the entities that refer to a goal are shown in the goal editor's "Referring Entities" table shown in Figure 43.

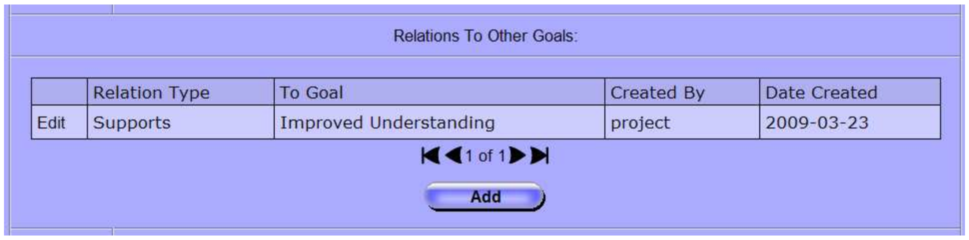

Referring Entities:

	Description	Created By	Date Created
Edit	Project: Requel	rreganjr	2009-01-04
Edit	Stakeholder: Harvard University	rreganjr	2009-01-05
Edit	UseCase: A user logs in to the system	rreganjr	2009-01-20

1 of 1

Figure 43: Goal Referring Entities

Upon saving the goal, the system initiates analysis of the goal as a background task. The assistant adds notes and issues to the goal as it analyzes it. During analysis of a goal the assistant identifies unknown or misspelled words; potential "significant terms" to add to the glossary; potential actors that don't already exist in the project; and complex sentences that may be difficult to understand. You may work on any requirements elements as the assistant analyzes them; the assistant will not interfere with your work.


Relations To Other Goals:

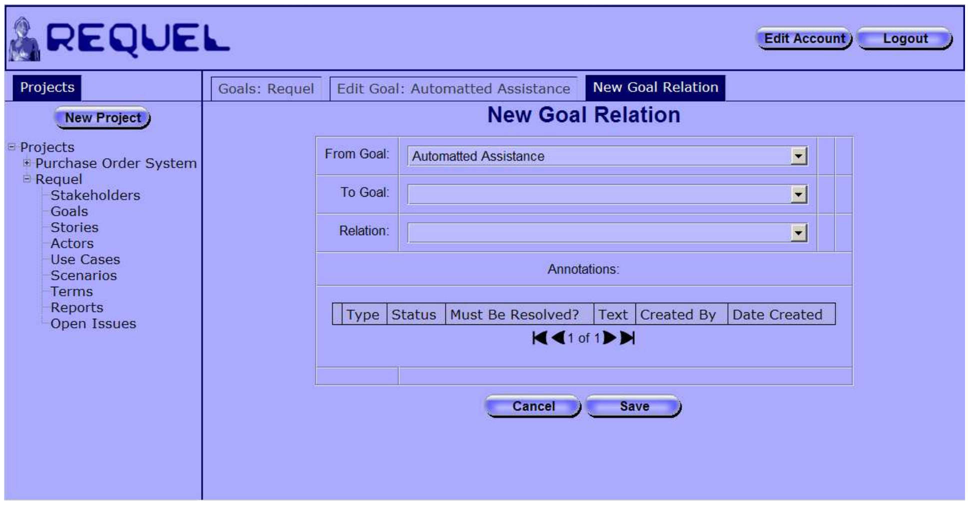
	Relation Type	To Goal	Created By	Date Created
Edit	Supports	Improved Understanding	project	2009-03-23

1 of 1

Add

Figure 44: Goal Relations

Goals can have relationships with other goals listed on the goal editor as shown in Figure 44. The relationships add traceability of goals from high level soft goals to specific functional requirements and as an indication of how goals interact with each other. Goals may support or conflict with each other. Conflicts help to identify places where tradeoffs may be needed, and to document the rationale behind the decisions made. Goal relations are directional, for example if goal “A” supports goal “B” there is a relationship from goal “A” to goal “B”, but not from “B” to “A”.

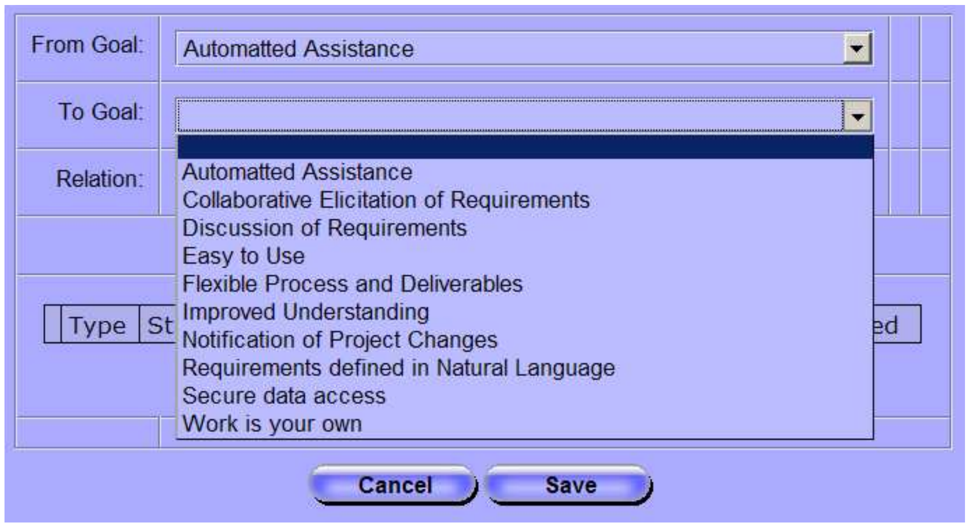


From Goal:	Automatted Assistance
To Goal:	
Relation:	

Type	Status	Must Be Resolved?	Text	Created By	Date Created

Figure 45: Goal Relation Editor Screen

Clicking the “Add” button opens the “New Goal Relation” editor as shown in Figure 45. A goal relation has a “From Goal”, a “To Goal”, and the relation type. The editor defaults the “From Goal” selector to the goal in the editor where the add button was clicked. Both the “To” and “From” goal fields list all the goals in the project as possible options as shown in Figure 46. After selecting the “To Goal” and relation type, clicking save creates the relationship and adds it to the goal relations table of the original goal as shown in Figure 44.


From Goal: Automatted Assistance

To Goal:

Relation:

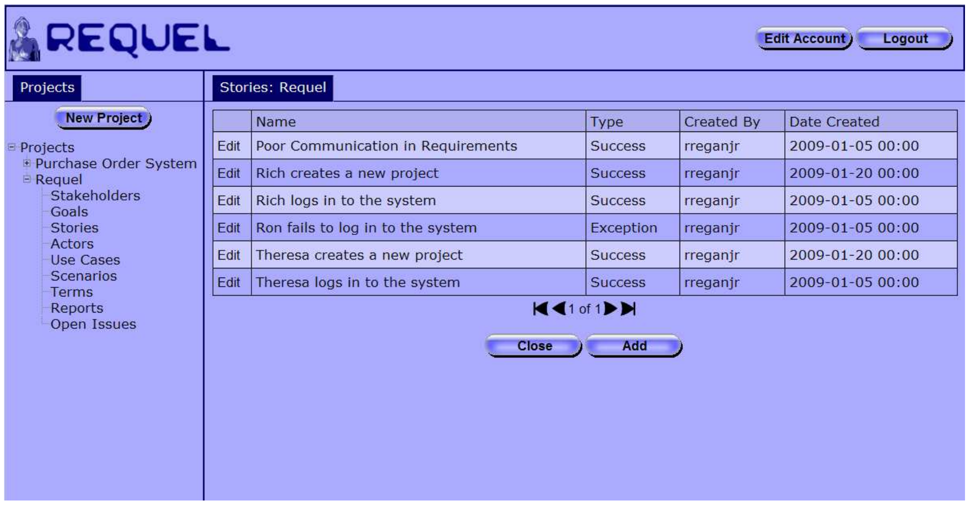
Automatted Assistance
Collaborative Elicitation of Requirements
Discussion of Requirements
Easy to Use
Flexible Process and Deliverables
Improved Understanding
Notification of Project Changes
Requirements defined in Natural Language
Secure data access
Work is your own

Figure 46: Goal Relation "To Goal" Selection

Creating and Editing Stories

Stories are simple narratives of how a user interacts with the system. Stories can be used to describe how things work before the new system is in place, i.e. the current state of affairs, or how things will be when the new system is in place. Stories can describe negative cases, such as problems with the current system, or how the new system should behave in a bad situation.

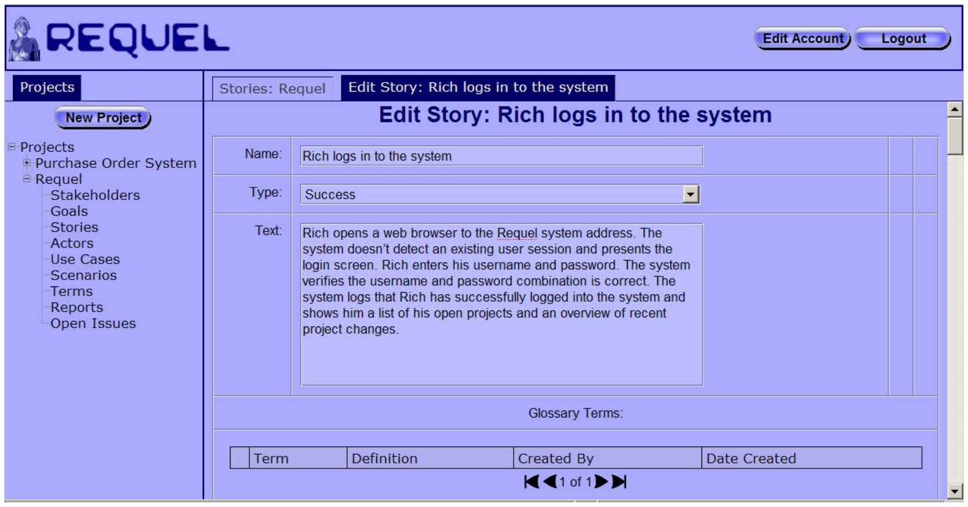
Clicking the “Stories” link in the project navigator tab opens the stories navigation screen as shown in Figure 47.



	Name	Type	Created By	Date Created
Edit	Poor Communication in Requirements	Success	rreganjr	2009-01-05 00:00
Edit	Rich creates a new project	Success	rreganjr	2009-01-20 00:00
Edit	Rich logs in to the system	Success	rreganjr	2009-01-05 00:00
Edit	Ron fails to log in to the system	Exception	rreganjr	2009-01-05 00:00
Edit	Theresa creates a new project	Success	rreganjr	2009-01-20 00:00
Edit	Theresa logs in to the system	Success	rreganjr	2009-01-05 00:00

Figure 47: Stories Navigation Screen

The stories navigator has a table with the name, type, who created it, and when it was created. If you are authorized to edit stories the first column contains an “Edit” link and below the table is an “Add” button. If you are not authorized to edit stories the first column contains a “View” link that opens the story editor in read-only mode.



Edit Story: Rich logs in to the system

Name: Rich logs in to the system

Type: Success

Text: Rich opens a web browser to the Requel system address. The system doesn't detect an existing user session and presents the login screen. Rich enters his username and password. The system verifies the username and password combination is correct. The system logs that Rich has successfully logged into the system and shows him a list of his open projects and an overview of recent project changes.

Glossary Terms:

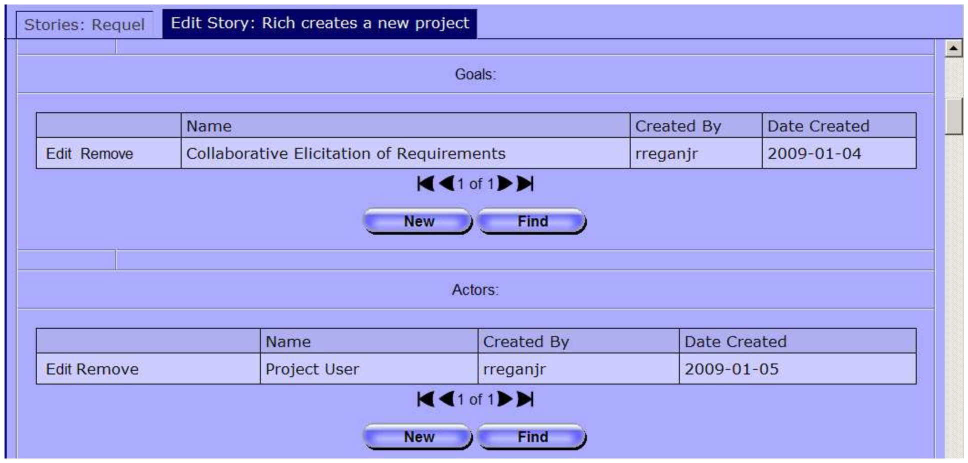
	Term	Definition	Created By	Date Created

Figure 48: Story Editor Screen

Clicking the “Edit” button for an existing story opens the story editor screen shown in Figure 48. A story has a short one line name, a type of “Success” or “Exception” and a text body. The story name must be unique in the project. Clicking the

"Save" button, located at the bottom of the screen, validates and creates or updates the story and initiates analysis as a background task.

Stories can optionally have goals and actors assigned. The actors add traceability to the types of users appropriate to the story. The goals can be used to indicate the goals supported by the story or derived from the story. For both goals and actors there are "New" and "Find" buttons as shown in Figure 49. The "New" button is used to open an editor for creating a new actor or goal that gets assigned to the story. The "Find" button is used to open a selector, like the one in Figure 40, to choose an existing goal or actor.



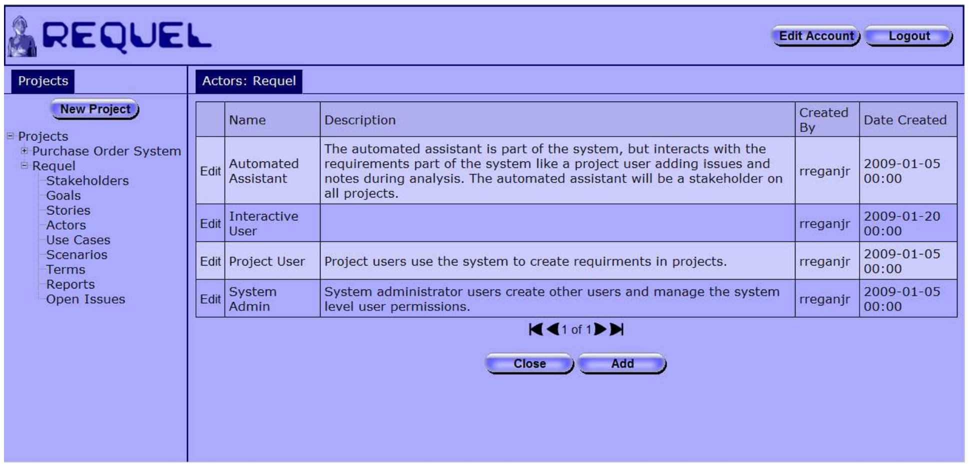
	Name	Created By	Date Created
Edit Remove	Collaborative Elicitation of Requirements	rreganjr	2009-01-04

	Name	Created By	Date Created
Edit Remove	Project User	rreganjr	2009-01-05

Figure 49: Story Goals and Actors

Creating and Editing Actors

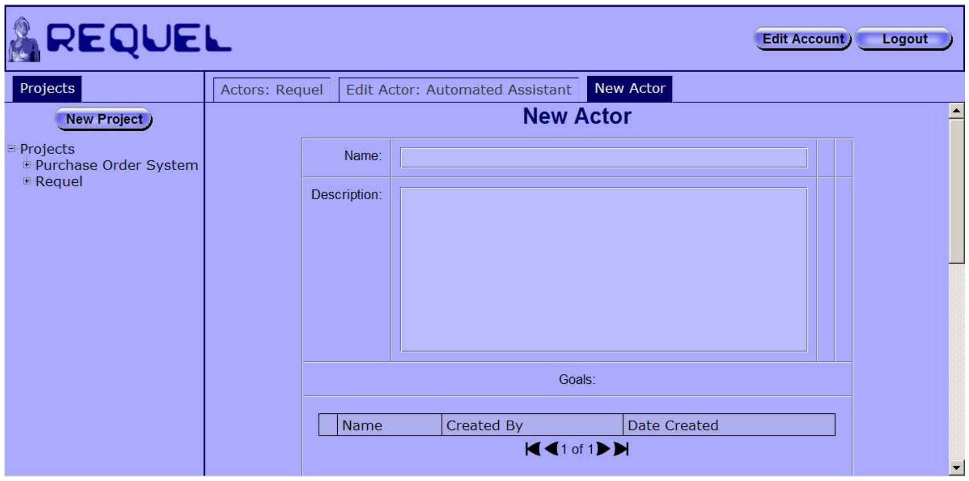
Actors represent the types of users that will use the system. Actors may be human users or other systems. Clicking on the "Actors" link in the project navigation tab opens the actor navigator screen shown in Figure 50. The actor navigator contains a table of the actors with columns for the name, description, who created it, and when it was created. If you are authorized to edit actors the first column contains an "Edit" link and below the table is an “Add” button. If you are not authorized to edit actors the first column contains a “View” link that opens the actor editor for read-only viewing.



	Name	Description	Created By	Date Created
Edit	Automated Assistant	The automated assistant is part of the system, but interacts with the requirements part of the system like a project user adding issues and notes during analysis. The automated assistant will be a stakeholder on all projects.	rreganjr	2009-01-05 00:00
Edit	Interactive User		rreganjr	2009-01-20 00:00
Edit	Project User	Project users use the system to create requirments in projects.	rreganjr	2009-01-05 00:00
Edit	System Admin	System administrator users create other users and manage the system level user permissions.	rreganjr	2009-01-05 00:00

Figure 50: Actors Navigator Screen

Clicking the “Add” button opens the “New Actor” editor as shown in Figure 51.



New Actor

Name:

Description:

Goals:

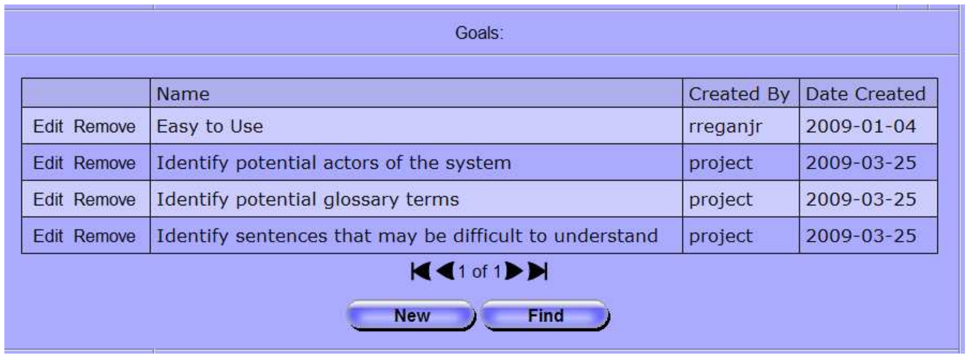
	Name	Created By	Date Created

Figure 51: New Actor Screen

An actor has a name that must be unique in the project, and a description. The description can be used to describe the actor’s primary function or to describe an example “persona” for a person or system that typifies the actor. For example the following is a persona for a user of the Requel system:

> Theresa is an accountant for Bailey Pet Supply tasked with specifying the requirements for integrating the purchase order system with the accounting system using the system. Theresa has never used a requirements tool before and has no projects. Theresa will be an end-user of the resulting system.

Actors may have goals that identify the functions of the system relevant to the actor, or general interests that the actor has that the system must meet. Figure 52 shows the goal references of an actor in the Requel system.


Goals:

	Name	Created By	Date Created
Edit Remove	Easy to Use	rreganjr	2009-01-04
Edit Remove	Identify potential actors of the system	project	2009-03-25
Edit Remove	Identify potential glossary terms	project	2009-03-25
Edit Remove	Identify sentences that may be difficult to understand	project	2009-03-25

Figure 52: Referenced Goals

From the actor editor new goals can be added using the "New" button or existing goals can be added using the "Find" button. Goals attached to the actor can be removed by clicking the "Remove" link in the first column of the goals table.

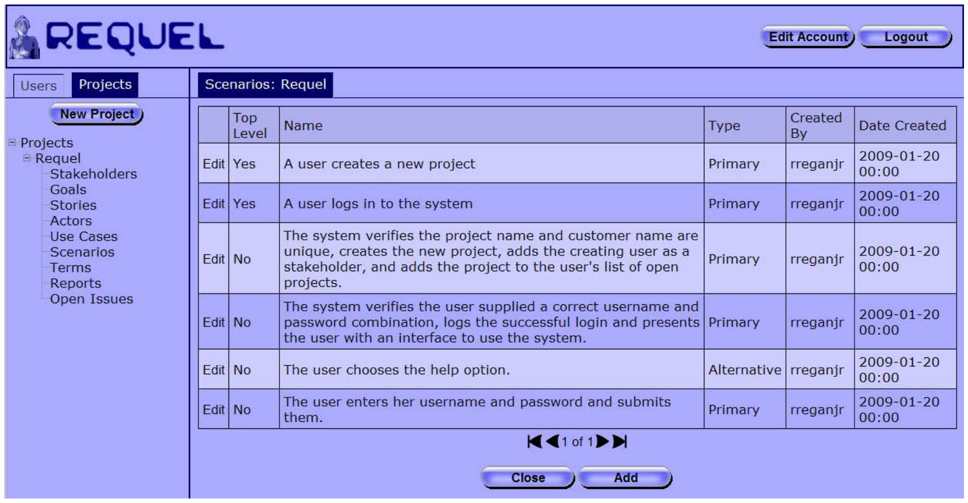
Like other requirements element actors are analyzed when saved.

Creating and Editing Scenarios

A scenario defines an interaction between an actor and the system. Scenarios are structured as a sequence of steps where the actor invokes an action on the system in one step, followed by a response by the system in the next step. A step may have sub-steps and any step with sub-steps is also a scenario in the system.

Clicking on the “Scenarios” link in the project navigator opens the scenarios navigator screen as shown in Figure 53. In addition to the standard name, created by, and date created columns, the scenario navigator has two additional columns: “Top Level” and “Type”. The top level column indicates if a scenario is used as a step in another scenario. Scenarios with a “Yes” value means the scenario is a top level scenario and not used in other scenarios. The type column indicates how the scenario is used, for example if it is used as an alternate, exceptional, or optional sequence of steps in another scenario.

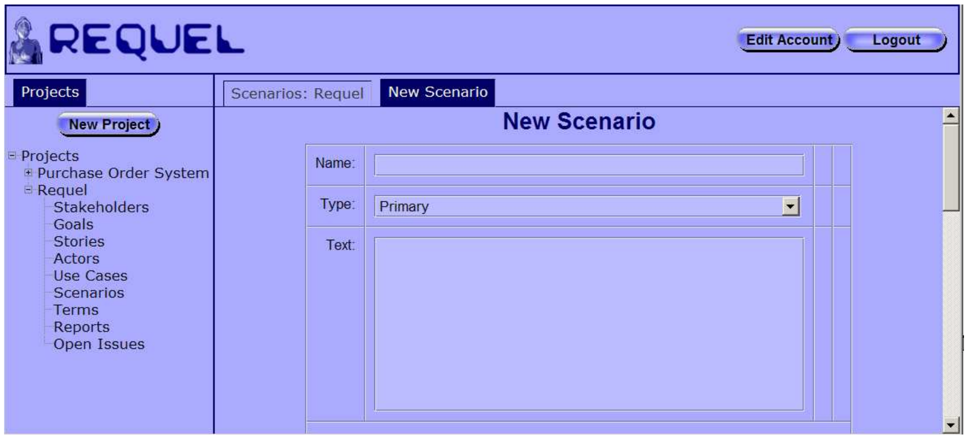
If you are authorized to edit scenarios the first column contains an “Edit” link and below the table is an “Add” button. If you are not authorized to edit scenarios the first column contains a “View” link that opens the scenario editor for read-only viewing.



	Top Level	Name	Type	Created By	Date Created
Edit	Yes	A user creates a new project	Primary	rreganjr	2009-01-20 00:00
Edit	Yes	A user logs in to the system	Primary	rreganjr	2009-01-20 00:00
Edit	No	The system verifies the project name and customer name are unique, creates the new project, adds the creating user as a stakeholder, and adds the project to the user's list of open projects.	Primary	rreganjr	2009-01-20 00:00
Edit	No	The system verifies the user supplied a correct username and password combination, logs the successful login and presents the user with an interface to use the system.	Primary	rreganjr	2009-01-20 00:00
Edit	No	The user chooses the help option.	Alternative	rreganjr	2009-01-20 00:00
Edit	No	The user enters her username and password and submits them.	Primary	rreganjr	2009-01-20 00:00

Figure 53: Scenarios Navigation Screen

Clicking the “Add” button opens the “New Scenario” editor as shown in Figure 54. A scenario has a name, type, optional descriptive text, and a sequence of steps. The scenario name must be unique in a project. The type of scenario is most relevant for scenarios that are steps in other scenarios and is described in the Editing Scenario Steps section, which gives a description of how to add and edit the steps of a scenario.

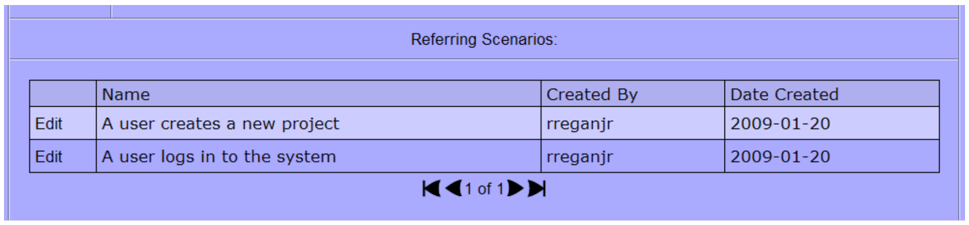


New Scenario

Name:	
Type:	Primary
Text:	

Figure 54: New Scenario Screen

The scenario editor lists the scenarios that use the scenario being edited in the "Referring Scenarios" table as shown in Figure 55. Clicking the edit link in the table opens the scenario editor for that scenario.



Referring Scenarios:

	Name	Created By	Date Created
Edit	A user creates a new project	rreganjr	2009-01-20
Edit	A user logs in to the system	rreganjr	2009-01-20

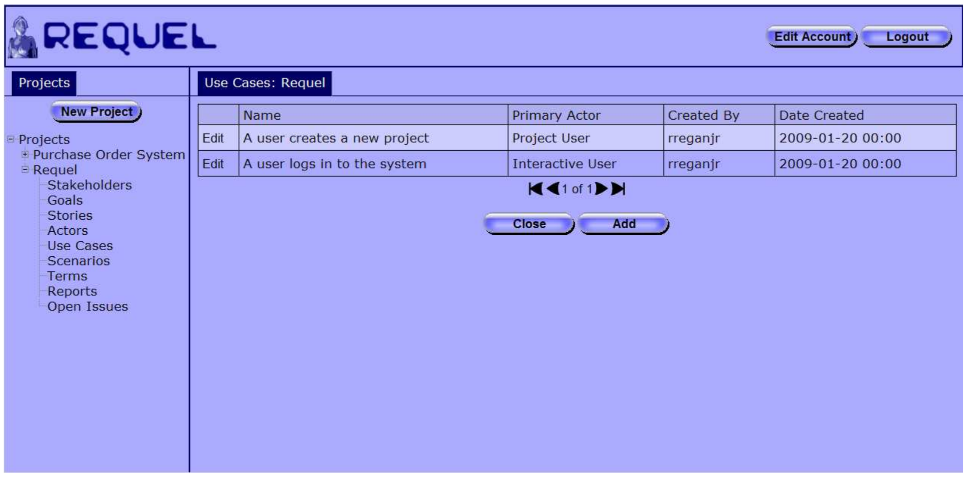
Figure 55: Scenario's Referring Scenarios

Creating and Editing Use Cases

Use cases represent functional requirements in terms of how a user interacts with the system. In the Requel system a use case is a confluence of stories, goals, and actors with a scenario to describe the behavior of a specific function of the system.

Clicking on the "Use Cases" link in the project navigator opens the use case navigator screen as shown in Figure 56. The navigator includes columns for the use case name, primary actor, created by username, and date created. If you are authorized to edit use cases the first column contains an "Edit" link and below the table is an "Add" button.

If you are not authorized to edit use cases the first column contains a "View" link that opens the use case editor for read-only viewing.



	Name	Primary Actor	Created By	Date Created
Edit	A user creates a new project	Project User	rreganjr	2009-01-20 00:00
Edit	A user logs in to the system	Interactive User	rreganjr	2009-01-20 00:00

Figure 56: Use Case Navigation Screen

Clicking the "Add" button opens the "New Use Case" editor as shown in Figure 57. A use case has a name, which must be unique in a project, an optional description, and a primary actor. The primary actor input is a combo-box where the actor can be selected from the drop-down list, or a name can be entered as text and a new actor is added to the project.

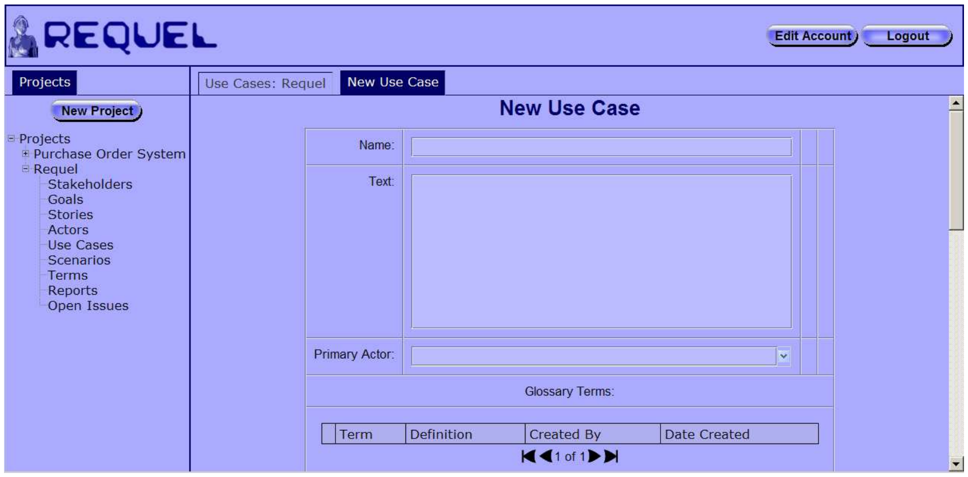
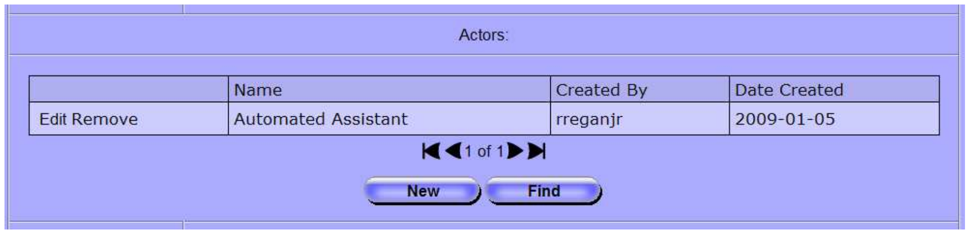


Figure 57: Use Case Edit Screen

A use case can contain a set of goals that represent what the primary actor is trying to accomplish with the use case. Goals are added and removed from use cases just like goals on other entities, such as actors as shown in Figure 52.

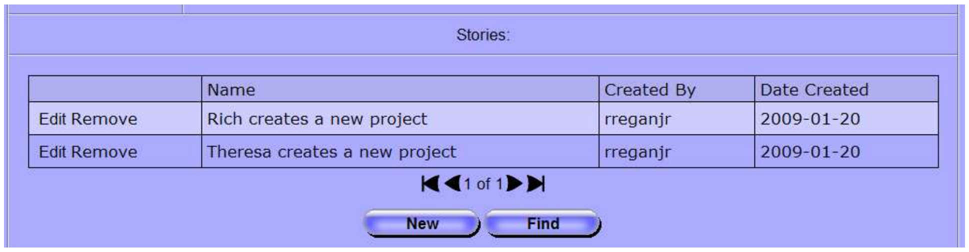
Most use cases only have a single primary actor, but in some cases additional actors are needed to show complex interactions where the system interacts with external services to support the user's actions. Like goals, from the use case editor a new actor can be created and added; or an existing actor can be referenced as shown in Figure 58.


Actors:

	Name	Created By	Date Created
Edit Remove	Automated Assistant	rreganjr	2009-01-05

Figure 58: Use Case Auxiliary Actors

Use cases can reference multiple stories as example of successful and exceptional interactions with the system. Stories are useful as rationale for understanding the scenario of the use case. Stories are displayed in a table as shown in Figure 59 and can be added and removed just like actors and goals.


Stories:

	Name	Created By	Date Created
Edit Remove	Rich creates a new project	rreganjr	2009-01-20
Edit Remove	Theresa creates a new project	rreganjr	2009-01-20

Figure 59: Use Case Stories

Editing the scenario of the use case is described in the Editing Scenario Steps section.

Editing Scenario Steps

The scenario step editor is used by both the scenario editor and use case editor screens. When creating a new scenario the steps editor appears with no steps and two buttons as shown in Figure 60.

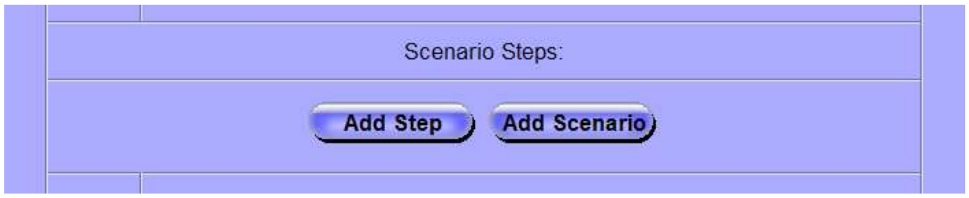


Figure 60: Empty Scenario Step Editor

The "Add Step" button creates an empty step node as shown in Figure 61. New steps are added below any existing steps.

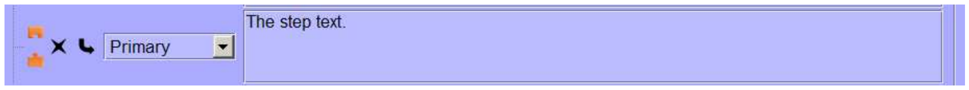


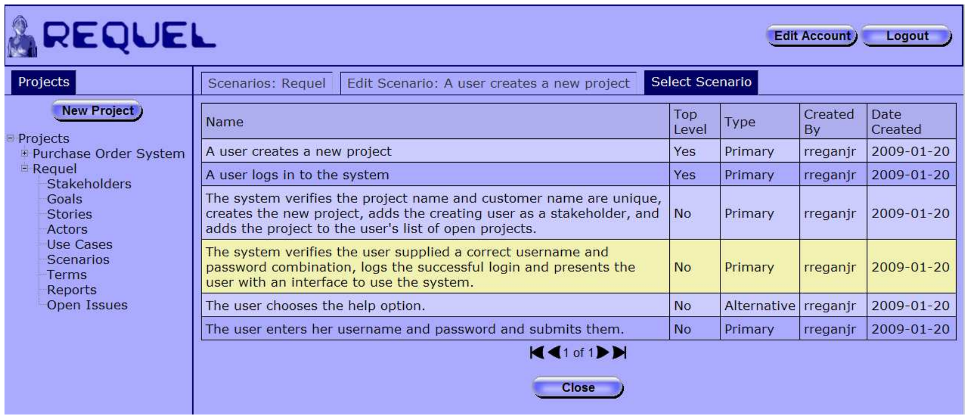
Figure 61: Empty Step Node

Each step has a box for editing the text, a drop down list for selecting the type, and a collection of buttons and icons for manipulating the step described in Table 25. The text of the step is limited to 255 characters. The types are pre-condition, primary, alternative, exception, and optional. A step with the pre-condition type isn't really a step, but indicates a condition that must be met for a scenario to start or continue. The primary type indicates the step is part of the primary flow of the scenario; alternative means the step is an alternative to the primary flow, but part of a successful interaction; exception indicates a problem or error condition; and optional means the step is a successful interaction, but one that is not required to take place.

🧩	The “slotted puzzle” icon is used to drag a step to a new location in the scenario by dropping it on the tabbed puzzle piece. A step can be dragged from and dropped to any sub-step level.
🧩	The “tabbed puzzle” icon is the drop target for moving a step. Dropping a step on the target inserts that step in the location and moves the existing all subsequent steps down by one.
✕	The “X” button removes the step from the scenario and moves all subsequent steps up by one. If the step is a simple step with no sub-steps it is deleted from the project. If the step is a sub-scenario it is removed, but not deleted.
↳	The “sub-step” button adds a new sub-step to this step. If the step has other sub-steps the new one is added at the end of the sequence.

Table 25: Step Node Controls

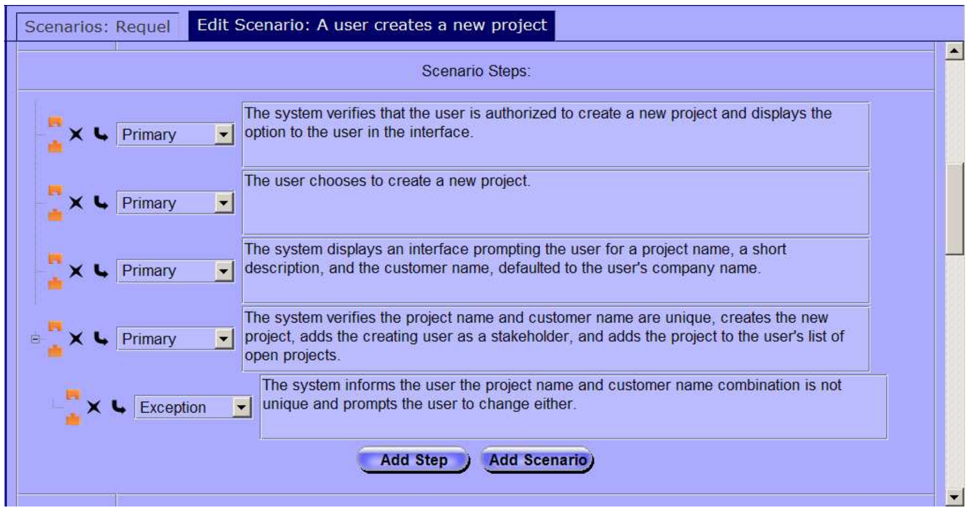
The “Add Scenario” button shown in Figure 60 is used to add an existing scenario as a step in the current scenario. Clicking the button opens the “Select Scenario” screen as shown in Figure 62. Clicking on a scenario selects it and adds it add as the last step in the scenario unless the original scenario is a step or sub-step reachable through the selected scenario, which would result in a cycle in the steps and is not permitted.



Name	Top Level	Type	Created By	Date Created
A user creates a new project	Yes	Primary	rreganjr	2009-01-20
A user logs in to the system	Yes	Primary	rreganjr	2009-01-20
The system verifies the project name and customer name are unique, creates the new project, adds the creating user as a stakeholder, and adds the project to the user's list of open projects.	No	Primary	rreganjr	2009-01-20
The system verifies the user supplied a correct username and password combination, logs the successful login and presents the user with an interface to use the system.	No	Primary	rreganjr	2009-01-20
The user chooses the help option.	No	Alternative	rreganjr	2009-01-20
The user enters her username and password and submits them.	No	Primary	rreganjr	2009-01-20

Figure 62: Select Scenario Screen

When editing a scenario with existing steps, the steps are ordered in the sequence they should occur with sub-steps below and to the right of their parent step as shown in Figure 63.



Primary — The system verifies that the user is authorized to create a new project and displays the option to the user in the interface.

Primary — The user chooses to create a new project.

Primary — The system displays an interface prompting the user for a project name, a short description, and the customer name, defaulted to the user's company name.

Primary — The system verifies the project name and customer name are unique, creates the new project, adds the creating user as a stakeholder, and adds the project to the user's list of open projects.

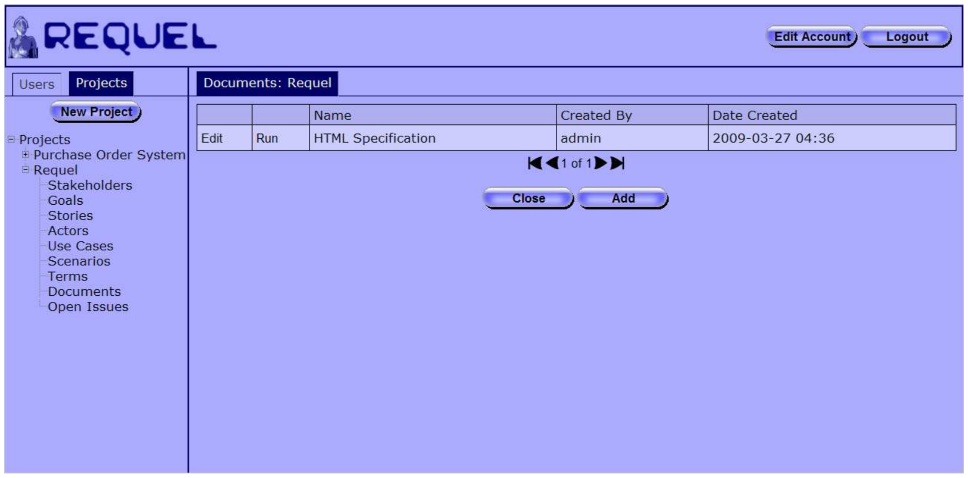
Exception — The system informs the user the project name and customer name combination is not unique and prompts the user to change either.

Figure 63: Scenario Steps Editor

Sub-steps can be nested to any level. Each step that contains sub-steps has a “-” button to the left of all other controls when the sub-steps are visible used to hide the sub-steps. When a step’s sub-steps are hidden the step has a “+” button that exposes the sub-steps when clicked.

Generating Documents

Once the requirements are in Requel you will want to generate a specification document to provide to the developers to build the system. Just as there are many processes for acquiring requirements, there are many formats for specifying requirements. Requel supports creating custom documentation using the XML Stylesheet Language Templates (XSLT) language to transform the requirements from an XML format into a specification document. XSLT is a powerful language that can transform XML into a variety of structured document formats such as HTML, PDF, and Microsoft Office documents using the Office Open XML format.

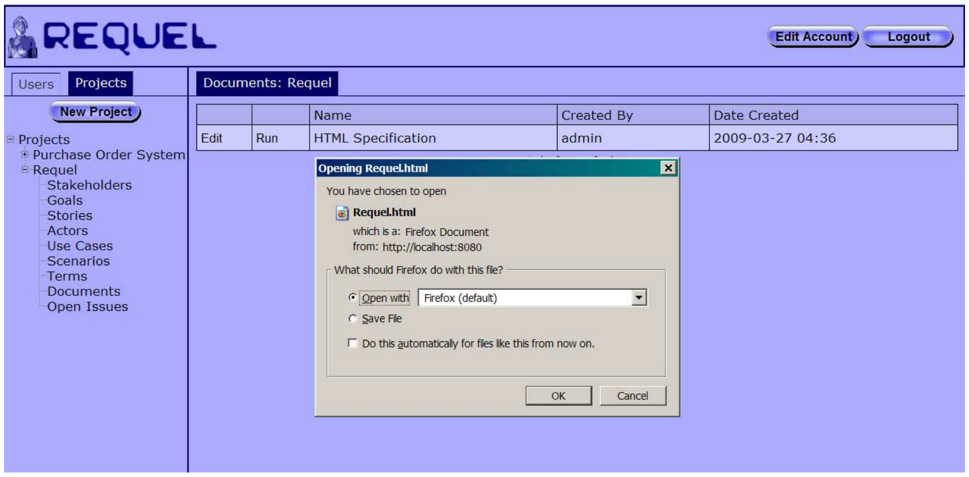


		Name	Created By	Date Created
Edit	Run	HTML Specification	admin	2009-03-27 04:36

Figure 64: Documents Navigator

Clicking on the “Documents” link in the project navigation opens the documents navigator as shown in Figure 64. Each entry in the table represents an XSLT script for generating a document. If you are authorized to edit documents (in the stakeholder permissions it is labeled as ReportGenerator) the first column contains an “Edit” link and below the table is an “Add” button. If you are not authorized to edit documents the first column contains a “View” link that opens the document editor for read-only viewing.

The second column of the table holds a “Run” link that executes the XSLT script on the requirements to generate a document. The document is returned to the browser, which gives you the option to open it or save it as shown in Figure 65. The document will have the project’s name with an appropriate extension for the type of document.



		Name	Created By	Date Created
Edit	Run	HTML Specification	admin	2009-03-27 04:36

Opening Requel.html

You have chosen to open

Requel.html

which is a: Firefox Document
from: http://localhost:8080

What should Firefox do with this file?

[✓ Open with] Firefox (default)

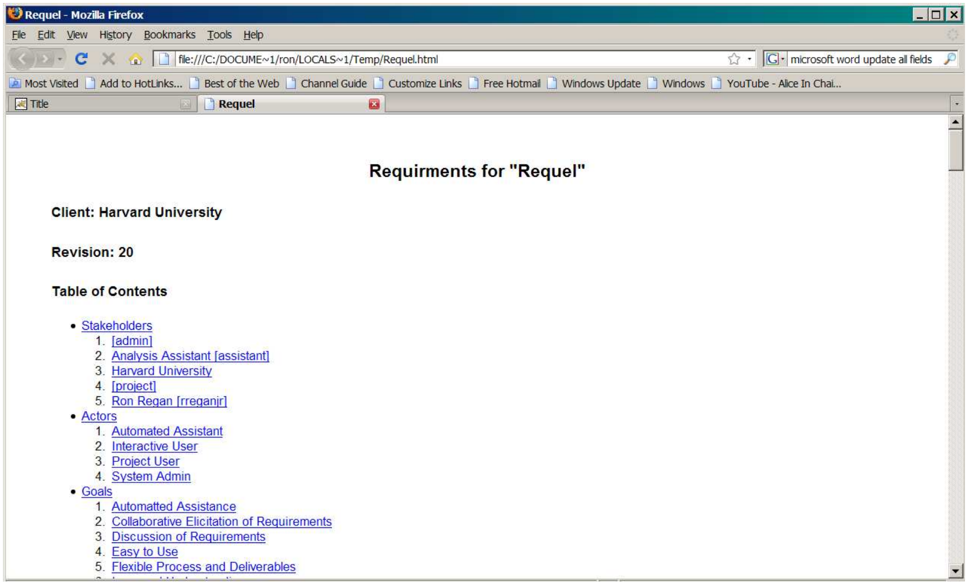
Save File

Do this automatically for files like this from now on.

OK Cancel

Figure 65: Generating a Document

The built-in "HTML Specification" document is added to all projects when they are created. It generates a simply formatted Web page with all requirements elements as shown in Figure 66. There is a section for each of the requirement element types: stakeholders, actors, goals, stories, use cases and scenarios. At the top of the document is a table of contents that links to each section and also to each element of the requirements.


Requel - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/DOCUME~1/ron/LOCALS~1/Temp/Requel.html
microsoft word update all fields
Most Visited
Add to HotLinks...
Best of the Web
Channel Guide
Customize Links
Free Hotmail
Windows Update
Windows
YouTube - Alice In Chai...
Title
Requel

Requirments for "Requel"

Client: Harvard University

Revision: 20

Table of Contents

- Stakeholders
  1. [admin]
  2. Analysis Assistant [assistant]
  3. Harvard University
  4. [project]
  5. Ron Regan [rreganjr]
- Actors
  1. Automated Assistant
  2. Interactive User
  3. Project User
  4. System Admin
- Goals
  1. Automatted Assistance
  2. Collaborative Elicitation of Requirements
  3. Discussion of Requirements
  4. Easy to Use
  5. Flexible Process and Deliverables

Figure 66: Requirements Document

Clicking the “Add” button opens the “New Use Case” editor as shown in Figure 67. A document has a name, which must be unique in the project, and the XSLT text. An XSLT can be uploaded from your PC to the system by clicking the “Browse” button, finding the file with the file upload dialog, and then clicking the “Upload” button. The contents of the XSLT will be loaded in the “Text” field. You can edit the XSLT in the text box, but it is not an effective tool for more than simple text changes.

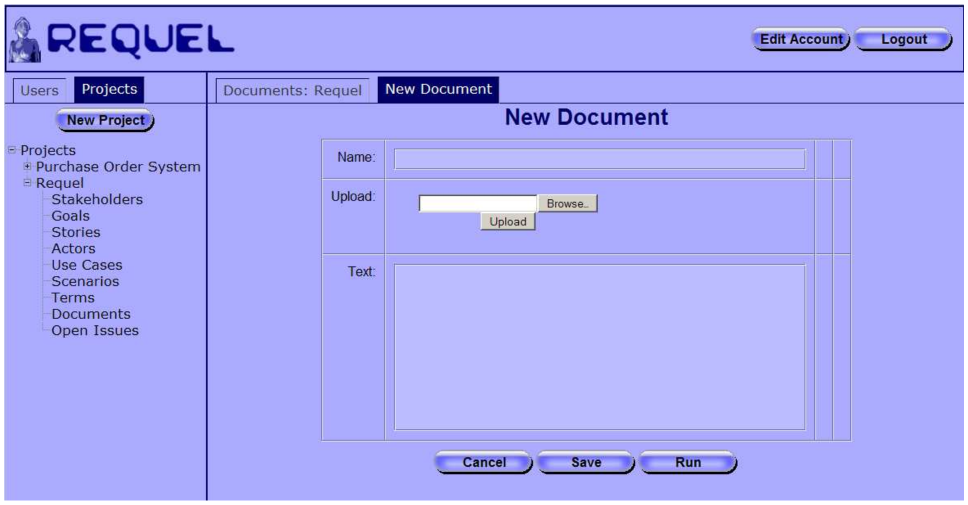
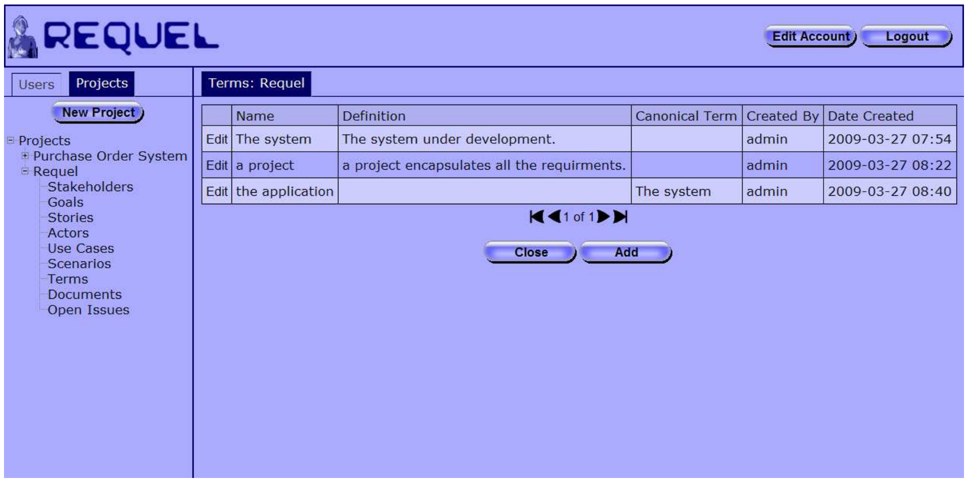


Figure 67: Document Editor Screen

Working with Terms

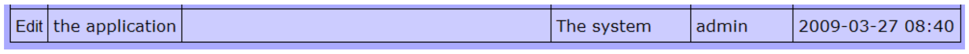
Each project has a glossary of terms. Using a glossary to explicitly define how words are used in a project can help improve understanding by using a consistent and explicitly defined set of terms. Clicking on the “Terms” link in the project navigator opens the terms navigator screen shown in Figure 68. If you are authorized to edit terms (in the stakeholder permissions it is labeled as GlossaryTerm) the first column contains an “Edit” link and below the table is an “Add” button. If you are not authorized to edit terms the first column contains a “View” link that opens the term editor for read-only viewing.



	Name	Definition	Canonical Term	Created By	Date Created
Edit	The system	The system under development.		admin	2009-03-27 07:54
Edit	a project	a project encapsulates all the requirments.		admin	2009-03-27 08:22
Edit	the application		The system	admin	2009-03-27 08:40

Figure 68: Terms Navigator

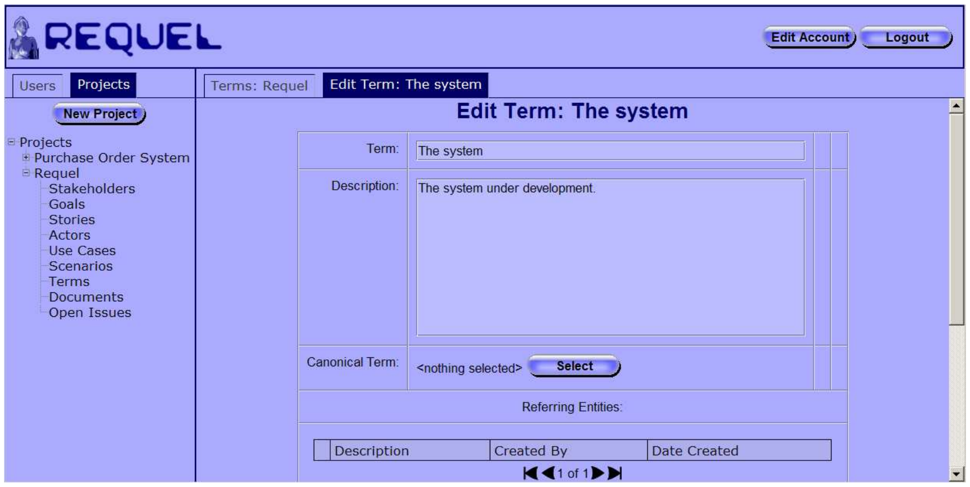
The terms navigator shows the text of the term in the "Name" column and the full text of the definition. The canonical term column contains the name of another term if the term is not the preferred term for the concept. For example the term "the application" in Figure 69 has the term "The system" as the canonical term. Ideally there should only be one term per concept and the system has a feature to solve this problem described below.



Edit	the application		The system	admin	2009-03-27 08:40

Figure 69: A Term with a Canonical Term

Clicking the "Edit" button on a term opens the term editor shown in Figure 70. The editor has text fields for the name and description and a selector button for choosing a canonical term.



Edit Term: The system

Term:	The system
Description:	The system under development.
Canonical Term:	<nothing selected> Select

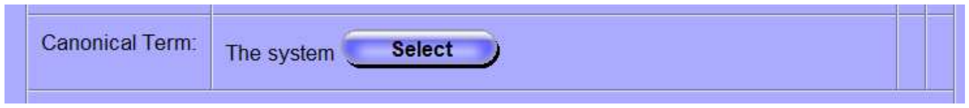
Referring Entities:

Description	Created By	Date Created

1 of 1

Figure 70: Term Editor

The selector button displays "<nothing selected>" when the term doesn't have a canonical term. Clicking the "Select" button opens a selector screen that looks like the navigator screen. When a canonical term is selected the name of the term shows up to the left of the button as shown in Figure 71. When creating a new term or changing the name of the term the system scans all the requirements and adds cross references between the elements and the term.



Canonical Term:	The system Select

Figure 71: Canonical Term Selected

As stated before, ideally there should only be one term per concept. In many cases different people use different terms to refer to the same thing when writing requirements individually. To solve the problem Requel has function that will replace all the occurrences of a term with its canonical term in all the requirement elements. First all the terms need to be added to the glossary. The assistant will probably create issues identifying them as potential glossary terms. Resolving the issues will create the terms and add references to all the requirement elements that use them. For all the terms that you want to replace, edit them and select the preferred term as the canonical term. At the bottom of the term editor is a “Replace Term” button shown in Figure 72. Clicking the button causes the system to update all the occurrences of the term in the requirements elements with its canonical term.

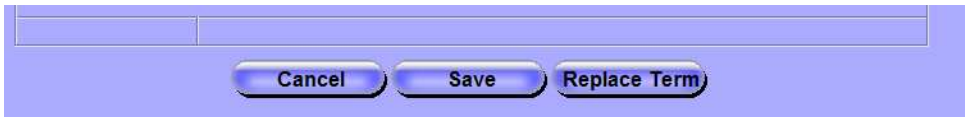
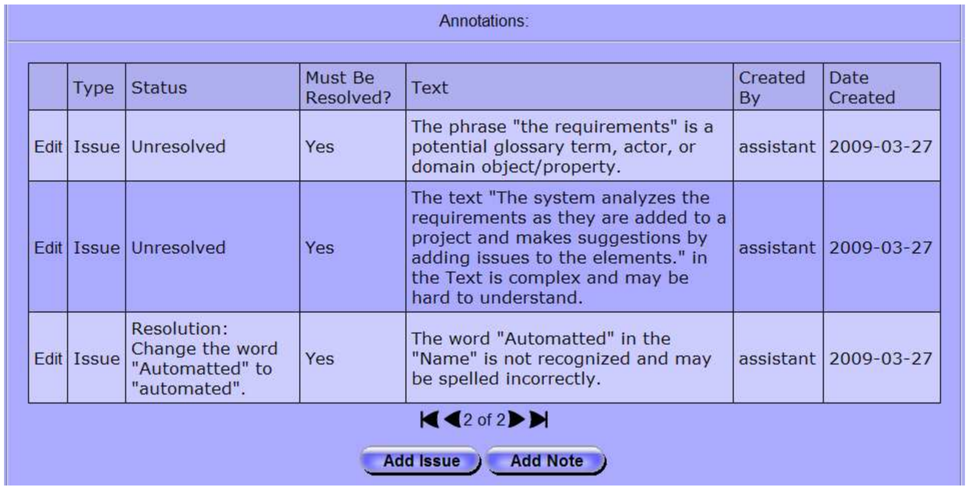


Figure 72: Replace Term

Adding Notes and Issues

All the requirements elements support adding notes and issues. Each of the requirements editors has an “Annotations” table with the notes and issues as shown in Figure 73. If you are authorized to edit annotations the first column contains an “Edit” link and below the table are “Add Issue” and “Add Note” buttons. If you are not authorized to edit annotations the first column contains a “View” link that opens the note or issue editor for read-only viewing.



Annotations:

	Type	Status	Must Be Resolved?	Text	Created By	Date Created
Edit	Issue	Unresolved	Yes	The phrase "the requirements" is a potential glossary term, actor, or domain object/property.	assistant	2009-03-27
Edit	Issue	Unresolved	Yes	The text "The system analyzes the requirements as they are added to a project and makes suggestions by adding issues to the elements." in the Text is complex and may be hard to understand.	assistant	2009-03-27
Edit	Issue	Resolution: Change the word "Automatted" to "automated".	Yes	The word "Automatted" in the "Name" is not recognized and may be spelled incorrectly.	assistant	2009-03-27

Figure 73: Annotations Section

The type column in the table indicates if the annotation is a note or issue. The status column will always have the value “Informational” for notes and either “Unresolved” or “Resolution: …” for issues. When an issue is resolved the status will start with “Resolution:” followed by the text of the position that resolved the issue. The “Must be Resolved?” column will always have “No” for notes and either “Yes” or “No” for issues. The “Text” column contains the full text of the issue.

Clicking on the “Add Note” button opens the “New Note” editor as shown in Figure 74. A note only has the text of the note.

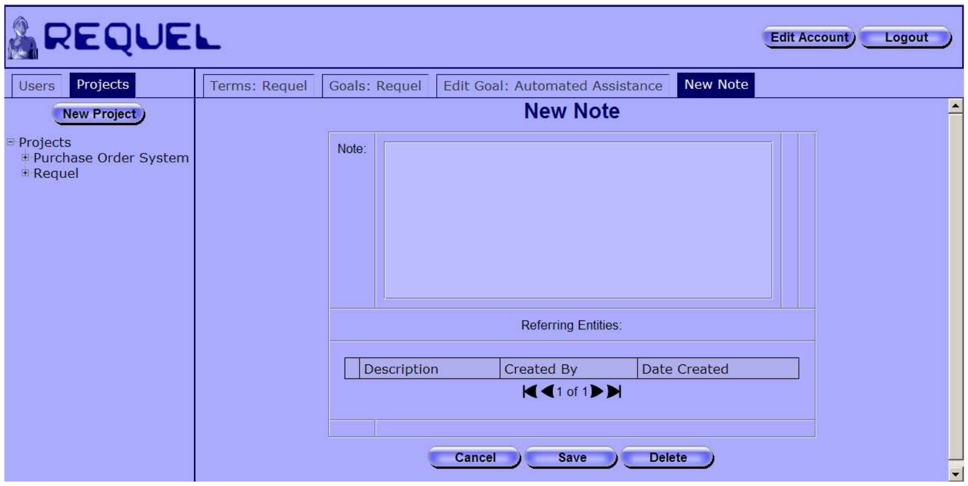
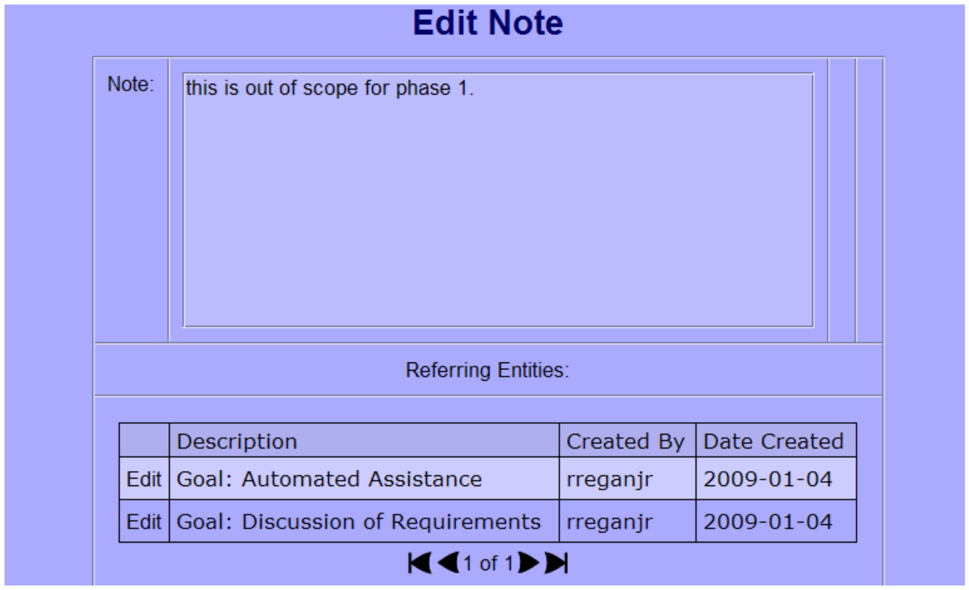


Figure 74: New Note Editor

There isn’t a way to find existing notes and add them to multiple requirement elements, but the system automatically detects adding notes with identical text and instead of creating a new note it will add the existing note to the requirement. Figure 75 shows a note shared by two goals, indicated in the “Referring Entities” table.



Edit Note

Note: this is out of scope for phase 1.

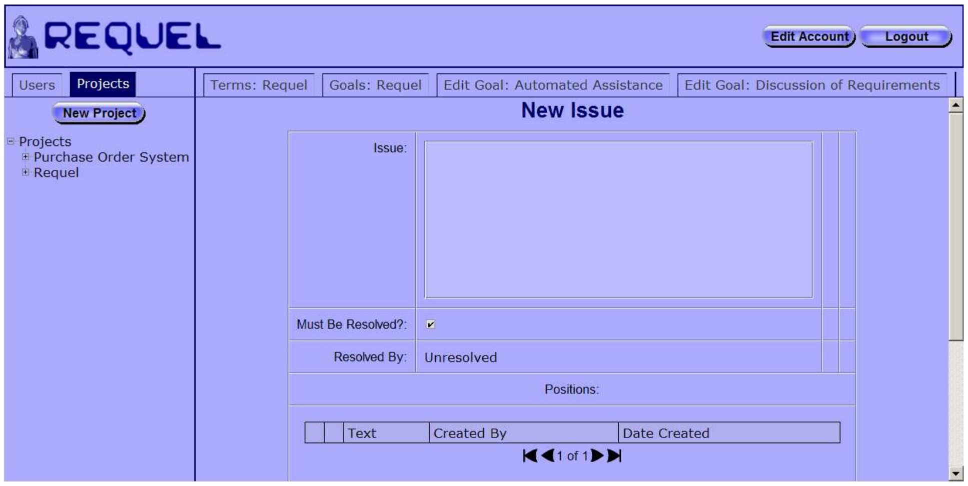
Referring Entities:

	Description	Created By	Date Created
Edit	Goal: Automated Assistance	rreganjr	2009-01-04
Edit	Goal: Discussion of Requirements	rreganjr	2009-01-04

1 of 1

Figure 75: Shared Notes

Clicking on the “Add Issue” button opens the “New Issue” editor as shown in Figure 76. An issue has text, a must be resolved checkbox, and positions. The must be resolved property is only for documentation purposes.



New Issue

Issue:

Must Be Resolved?: ✓

Resolved By: Unresolved

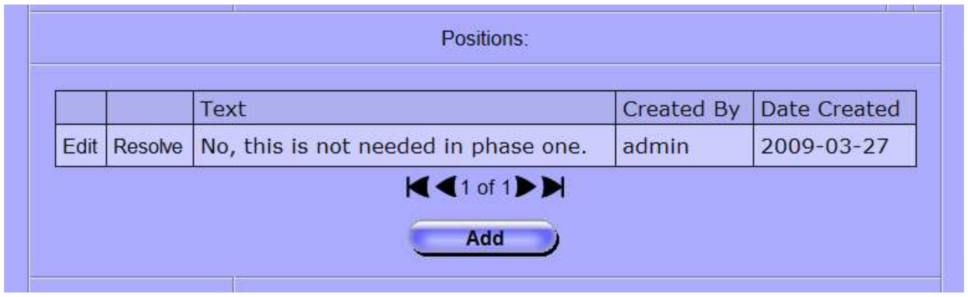
Positions:

		Text	Created By	Date Created

1 of 1

Figure 76: New Issue Editor

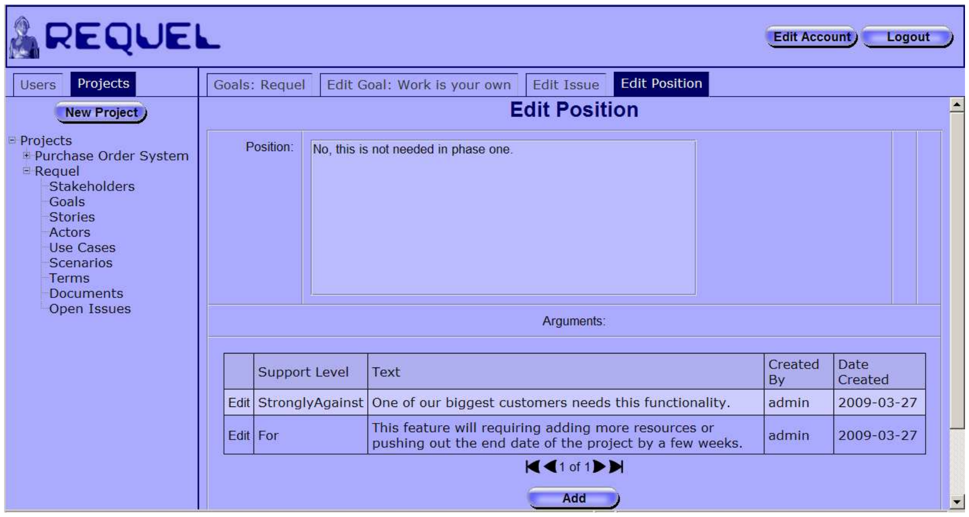
After an issue is saved positions can be added by users authorized to edit issues by clicking the “Add” button below the positions table as shown in Figure 77.



		Text	Created By	Date Created
Edit	Resolve	No, this is not needed in phase one.	admin	2009-03-27

Figure 77: Adding Positions

Positions represent potential solutions to an issue. The position editor has a text field to enter the text of the position and a table of arguments that argue for or against the position as a solution to the issue as shown in Figure 78. After a position is saved arguments can be added by users authorized to edit issues by clicking the "Add" button below the arguments table.



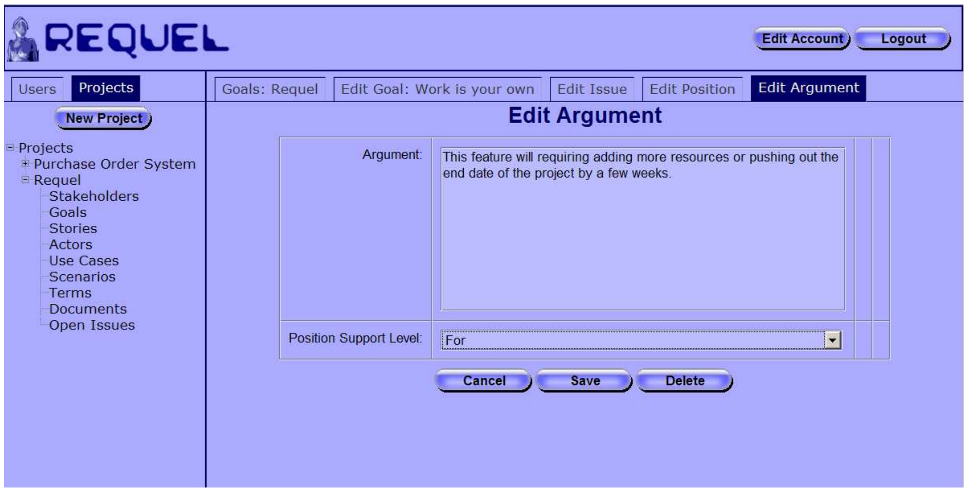
Position: No, this is not needed in phase one.

Arguments:

	Support Level	Text	Created By	Date Created
Edit	StronglyAgainst	One of our biggest customers needs this functionality.	admin	2009-03-27
Edit	For	This feature will requiring adding more resources or pushing out the end date of the project by a few weeks.	admin	2009-03-27

Figure 78: Position Editor

Arguments are the reasons why a particular position of an issue should or should not be chosen as the solution to an issue. Arguments have text description and a support level with values from "strongly against" to "strongly for." Clicking the add or edit button opens the argument editor shown in Figure 79. The argument editor has a field for entering the text of the argument and a drop down list with the support levels.



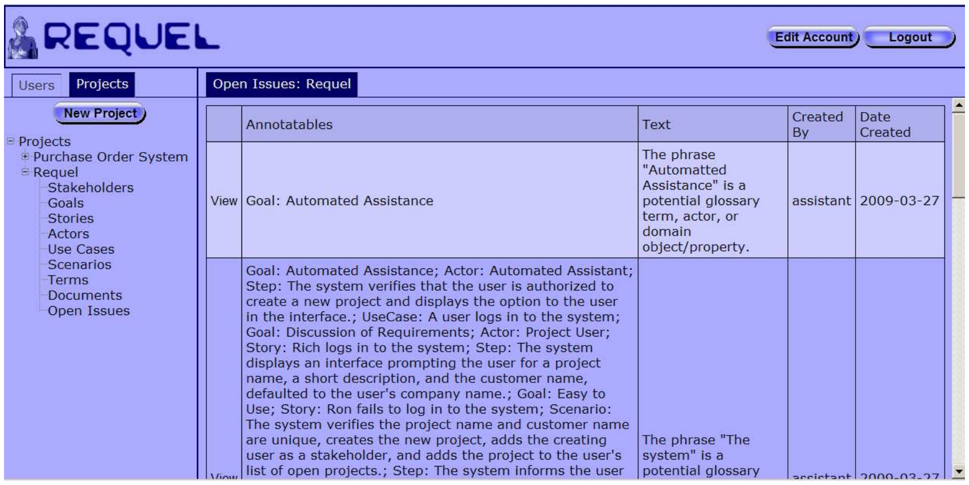
Edit Argument

Argument: This feature will requiring adding more resources or pushing out the end date of the project by a few weeks.

Position Support Level: For

Figure 79: Argument Editor

All the open issues for all the elements of a project can be seen on the "Open Issues" screen as shown in Figure 80. The screen can be opened from the project navigator "Open Issues" link. The issues screen has a "View" link for each issue that opens the issue editor. The issues table has a column labeled "Annotatables" that lists out all of the requirements elements that are assigned to the issue.

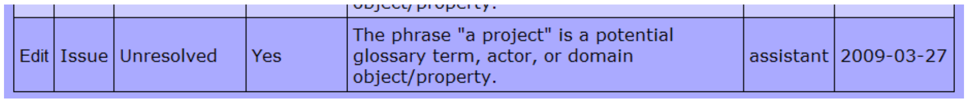


	Annotatables	Text	Created By	Date Created
View	Goal: Automated Assistance	The phrase "Automatted Assistance" is a potential glossary term, actor, or domain object/property.	assistant	2009-03-27
View	Goal: Automated Assistance; Actor: Automated Assistant; Step: The system verifies that the user is authorized to create a new project and displays the option to the user in the interface.; UseCase: A user logs in to the system; Goal: Discussion of Requirements; Actor: Project User; Story: Rich logs in to the system; Step: The system displays an interface prompting the user for a project name, a short description, and the customer name, defaulted to the user's company name.; Goal: Easy to Use; Story: Ron fails to log in to the system; Scenario: The system verifies the project name and customer name are unique, creates the new project, adds the creating user as a stakeholder, and adds the project to the user's list of open projects.; Step: The system informs the user	The phrase "The system" is a potential glossary	assistant	2009-03-27

Figure 80: Open Issues

Working with Analysis Issues

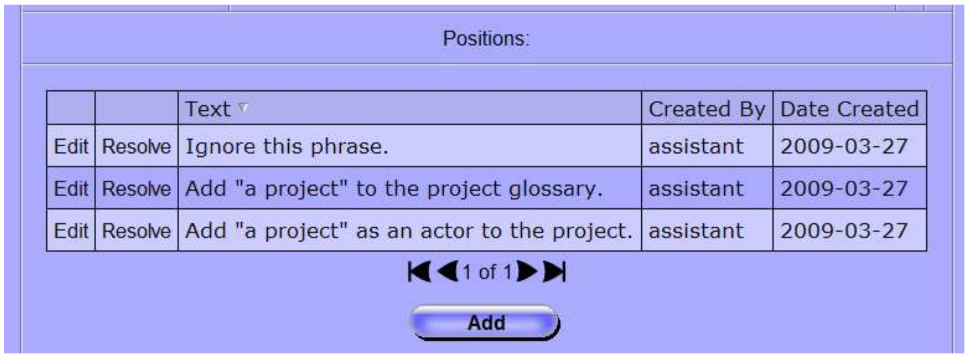
Every time a requirement element is saved the system analyses it and adds issues for potential problems or improvements the assistant detects. The issues will concern potential spelling errors or unfamiliar words, possible glossary terms and actors, complex sentences, and poorly structured scenario steps. Issues added during analysis will always have the "assistant" user as the creator as shown in Figure 81.



Edit	Issue	Unresolved	Yes	The phrase "a project" is a potential glossary term, actor, or domain object/property.	assistant	2009-03-27

Figure 81: Example Analysis Issue

The issues added by the assistant have specialized positions that automate the task describe in the position. Figure 82 shows the positions of the issue "The phrase 'a project' is a potential glossary term, actor, or domain object" from Figure 81. The position "Ignore this phrase" resolves the issue without taking any action. The position "Add 'a project' to the project glossary" adds the term "a project" to the glossary and a reference to all the project elements that are assigned to the issue when used to resolve the issue.



Positions:

		Text	Created By	Date Created
Edit	Resolve	Ignore this phrase.	assistant	2009-03-27
Edit	Resolve	Add "a project" to the project glossary.	assistant	2009-03-27
Edit	Resolve	Add "a project" as an actor to the project.	assistant	2009-03-27

Figure 82: Example Analysis Positions

Table 26 lists the text of the special positions that invoke an automated task when used to resolve an issue.

Position Text	**Action taken when the position is used to resolve the issue.**
Add "…" to the project glossary.	Creates the term in the glossary and adds a reference to all the project elements that are assigned to the issue.
Add "…" as an actor to the project.	Creates a new actor in the project and adds a reference to all the project elements that are assigned to the issue.
Change the word "…" to "…".	Updates the name and text of all the requirement elements that are assigned to the issue to replace the original text with the new text.
Add "…" to the dictionary.	Adds the word to the systems global dictionary so that future occurrences of the word will not be identified as unknown.
Add the actor "…" to the use case.	Adds the actor to the auxiliary actors of the use case.
Change "…" to the primary actor.	Replaces the text in the scenario step with the name of the primary actor.

Table 26: Automated Task Positions

Requel Setup

This section describes the setup of the Requel system. The system is distributed as a Java Web archive file and requires only minimal configuration to setup.

System Requirements

Requel is a Web application implemented in the Java language and based on the Java Enterprise Edition (JEE5) platform, intended to run in a standard Web container. Requel requires the standard Sun Microsystems Java runtime version 6 update 4 or later to operate properly. Requel has dependencies on Sun's virtual machine and will not work with a third-party virtual machine. Requel is known to work with versions 5.5 and 6.0 of the Apache Tomcat Web container.

Requel stores user and project data in a standard SQL database. It is known to work with MySQL Server version 5.0. The database initialization scripts may have MySQL specific syntax and commands and the configuration

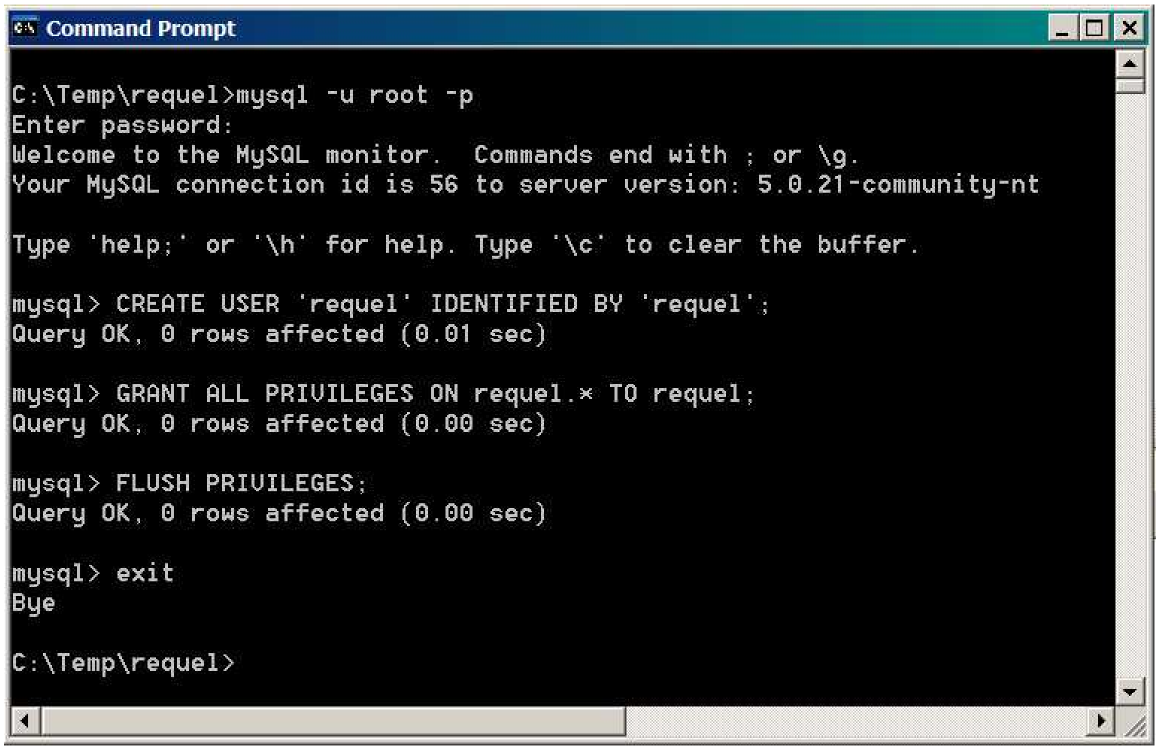
Create a Database User

Before Requel can be installed a database user account must be created. Requel will use this user account to create and initialize the database when first installed, and to update the requirements and user data during normal operation. It is important that the user has all privileges granted to the database name that Requel will use.

The MySQL commands below create a user named ‘requel’ with the password ‘password’ and grants all privileges to that user for all objects in the database named ‘requeldb’. The database should not be created ahead of time; Requel will create it on the first start.

```
CREATE USER 'requel' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON requeldb.* TO requel;
FLUSH PRIVILEGES;
```

Figure 83 shows how to use the MySQL command line interface to login as the root user and create the user. If the Web server and database server are on different physical servers, then in the create user and grant commands append the Web server hostname or IP address to the username. For example, requel@192.168.1.100.



```
C:\Temp\requel>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 56 to server version: 5.0.21-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE USER 'requel' IDENTIFIED BY 'requel';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON requel.* TO requel;
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye

C:\Temp\requel>
```

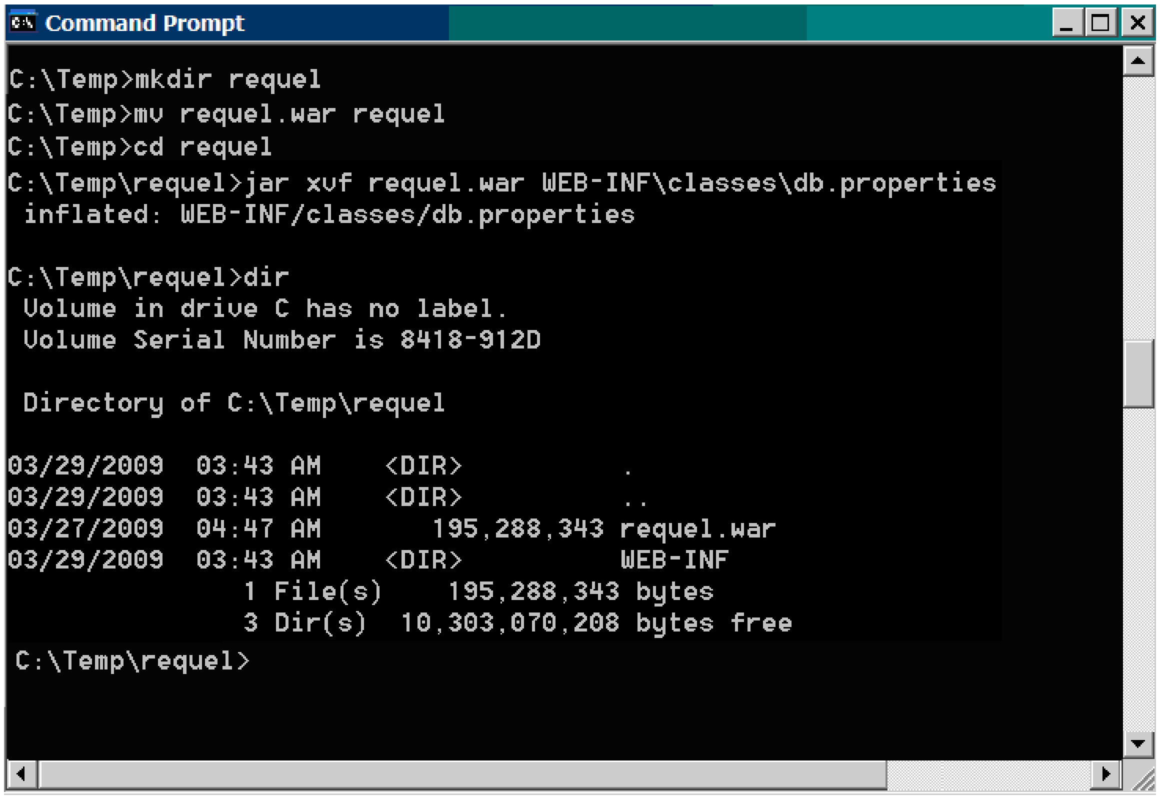
Figure 83: Create the Database User

Configuration

The Requel system requires minimal configuration to get running. The database settings are the only settings that must be configured before the system can be used and they should be configured before installing the system in a Web server.

To configure the database settings the db.properties file must be unpacked from the war file. This can be done using the Java jar command as shown in Figure 84. Create a directory to hold the contents of the war file and copy the war file into that directory. Use the jar command below to extract the file:

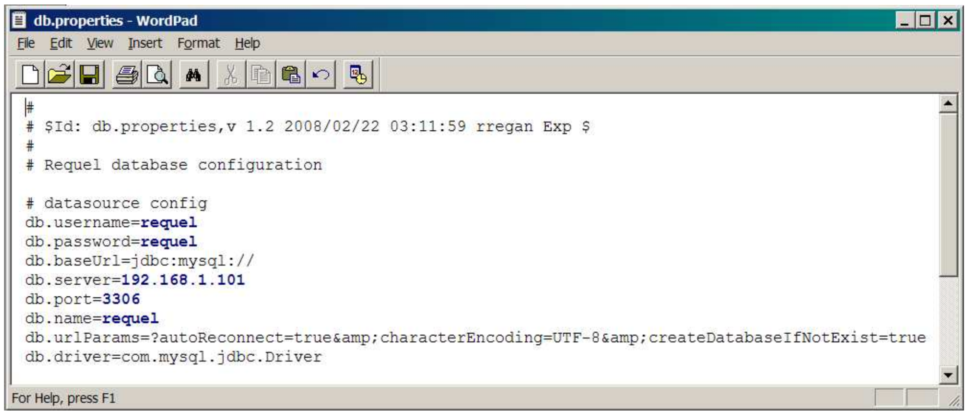
```
jar xvf requel.war WEB-INF\classes\db.properties
```



```
C:\Temp>mkdir requel
C:\Temp>mv requel.war requel
C:\Temp>cd requel
C:\Temp\requel>jar xvf requel.war WEB-INF\classes\db.properties
 inflated: WEB-INF/classes/db.properties

C:\Temp\requel>dir
 Volume in drive C has no label.
 Volume Serial Number is 8418-912D

 Directory of C:\Temp\requel

03/29/2009  03:43 AM    <DIR>          .
03/29/2009  03:43 AM    <DIR>          ..
03/27/2009  04:47 AM       195,288,343 requel.war
03/29/2009  03:43 AM    <DIR>          WEB-INF
               1 File(s)    195,288,343 bytes
               3 Dir(s)  10,303,070,208 bytes free
C:\Temp\requel>
```

Figure 84: Extracting the db.properties file from the WAR File

When the jar command completes there will be a WEB-INF directory. Open the file WEB-INF\classes\db.properties in a text editor such as WordPad in Windows as shown in Figure 85. You will need to set the db.username, db.password, db.server, db.port, and db.name properties. The database user must have database creation permission if the database specified in the db.name property doesn’t already exist.

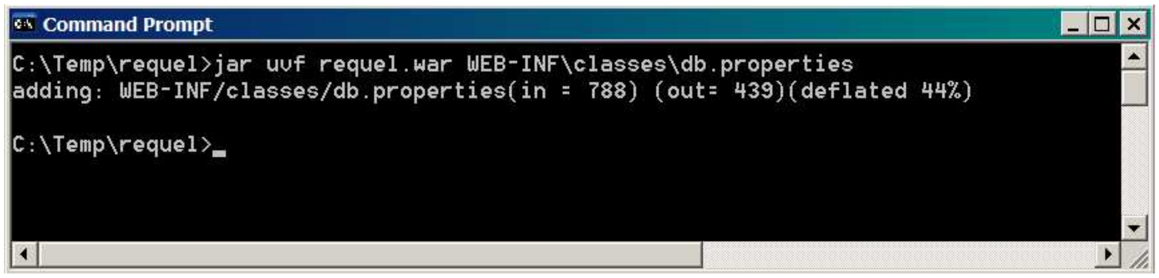


```
#
# $Id: db.properties,v 1.2 2008/02/22 03:11:59 rregan Exp $
#
# Requel database configuration

# datasource config
db.username=requel
db.password=requel
db.baseUrl=jdbc:mysql://
db.server=192.168.1.101
db.port=3306
db.name=requel
db.urlParams=?autoReconnect=true&amp;characterEncoding=UTF-8&amp;createDatabaseIfNotExist=true
db.driver=com.mysql.jdbc.Driver
```

Figure 85: Editing the Database Properties

After saving the properties file, it must be added back to the war file. Use the jar command below to add the updated properties file to the war file as shown in Figure 86.

```
jar uvf requel.war WEB-INF\classes\db.properties
```



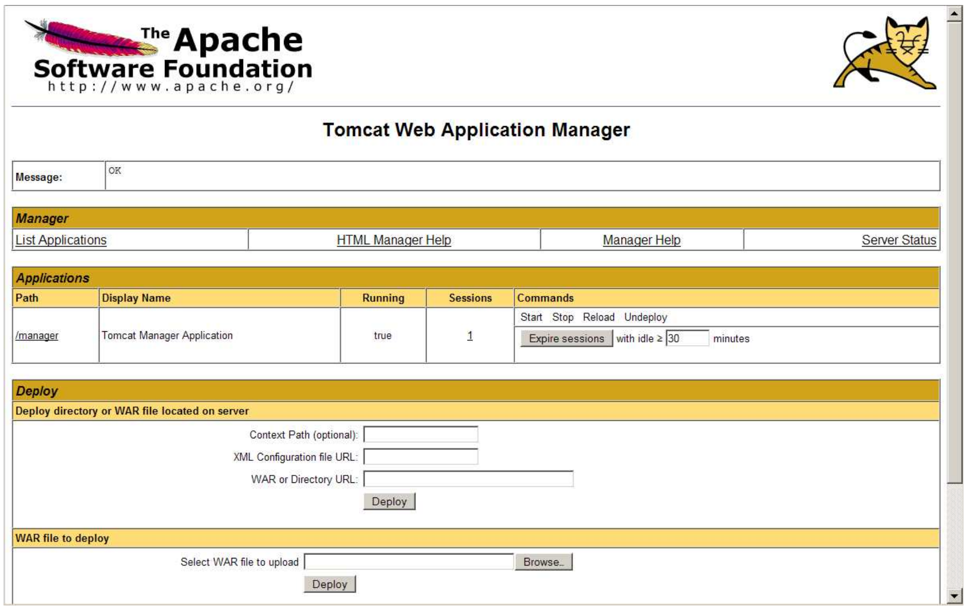
```
C:\Temp\requel>jar uvf requel.war WEB-INF\classes\db.properties
adding: WEB-INF/classes/db.properties(in = 788) (out= 439)(deflated 44%)

C:\Temp\requel>_
```

Figure 86: Updating the WAR file

The war file is now ready to be deployed to a Web server.

Deploying to Apache Tomcat

Tomcat is a freely available open source JEE Web container. Use the Web application manger that comes with Tomcat to deploy the Requel war. Open a Web browser to the manager page of the Tomcat server. The server prompts you for a username and password and then displays the manager page as shown in Figure 87.



The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

Manager

List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Display Name	Running	Sessions	Commands
/manager	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

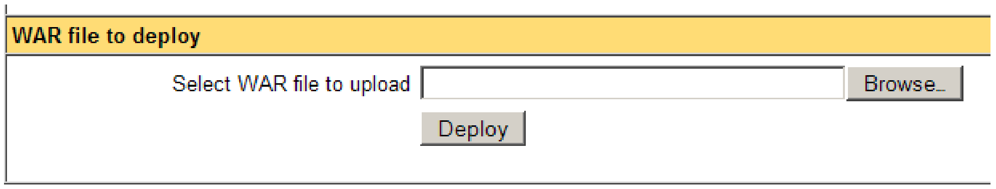
Context Path (optional):
XML Configuration file URL:
WAR or Directory URL:
Deploy

WAR file to deploy

Select WAR file to upload Browse...
Deploy

Figure 87: Apache Tomcat Manager

At the bottom of the page is the war file deployment form as shown in Figure 88. Use the browse button to locate the Requel war file with the updated database properties and then click the Deploy button to upload and install the application. It will typically take from between five and twenty minutes for Requel to create and initialize the database depending on the performance of the database server.

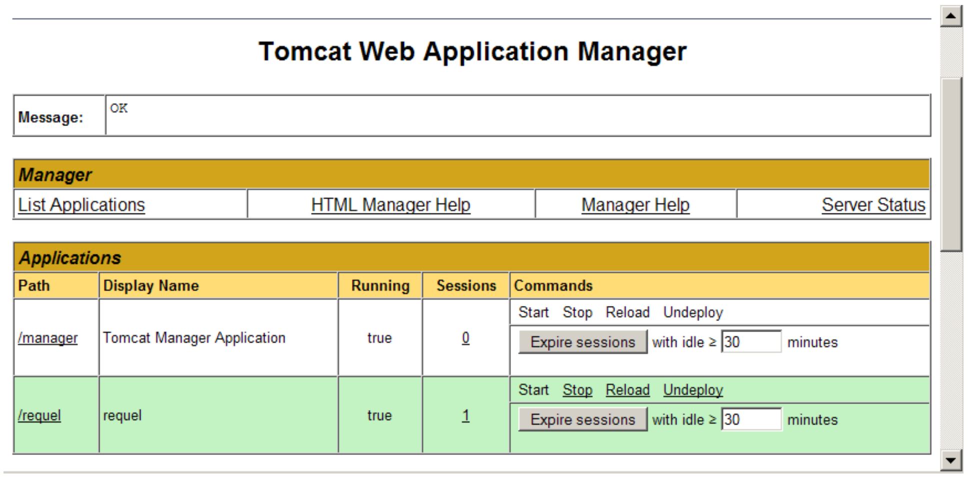


WAR file to deploy

Select WAR file to upload Browse...
Deploy

Figure 88: WAR File Deployment

When the deployment completes the application manager lists Requel in the applications section as shown in Figure 89. Clicking the "/requel" link in the "Path" column of the table connects to Requel and the login screen should appear as shown in Figure 26. You should record the URL from the address bar of the Web browser to send to users for accessing the system.



Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Display Name	Running	Sessions	Commands
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/requel	requel	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 89: Requel in the Tomcat Application Manager

Completing the Setup

The first thing to do when the system is ready for access is to login using the built-in administrator account and change the password. The username and password are both set to 'admin' when the system is initialized. See the User Administration section for instructions on changing the password. You should also change the password of the built-in project and assistant users.

The system is now ready to be used. The next step is creating accounts for the users to access the system. See the User Administration section for instructions on creating users.

User Administration

The primary task of system administrators is creating and managing users. System users with the system administrator role are authorized to edit user accounts for all users, and to add new users.

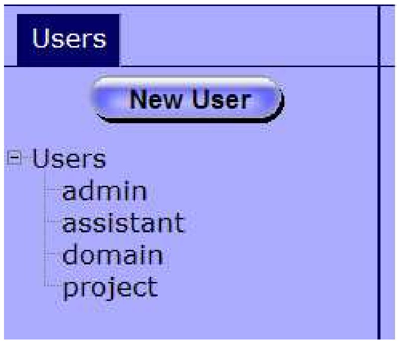


Figure 90: Users Navigation Tab

After logging in the left hand side of the screen will contain a tab labeled "Users" with a "New User" button for creating new users and tree containing all the users of the system as shown in Figure 90. Clicking the new user button opens the "New User" editor as shown in Figure 91. Clicking on a user name in the tree opens that user for editing. Each user must have a username, password, organization, email address, and at least one user role. The name and phone number are optional.

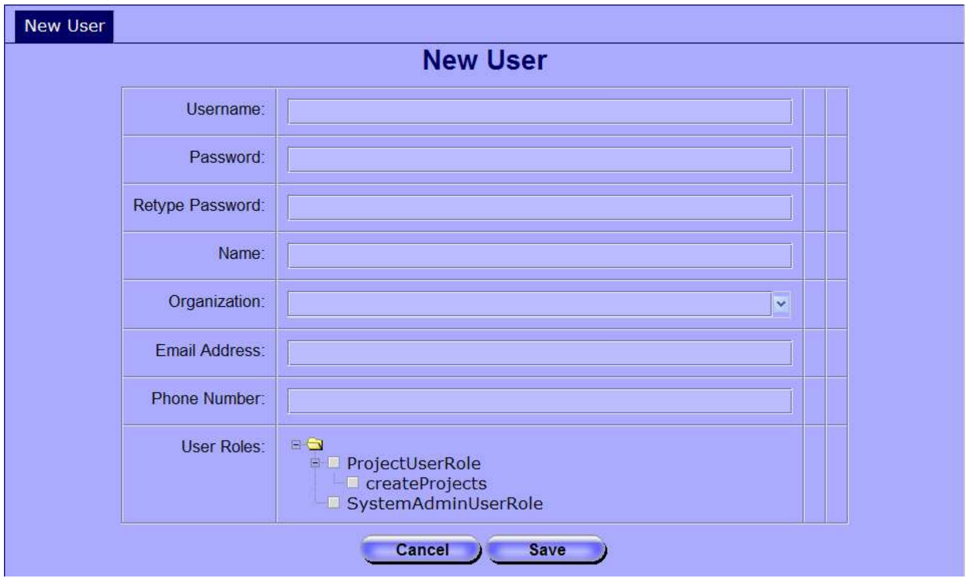
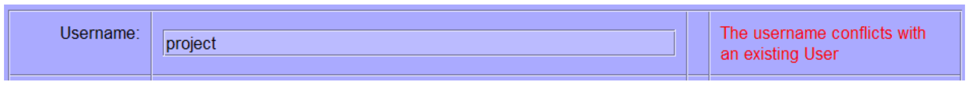


Figure 91: User Editor

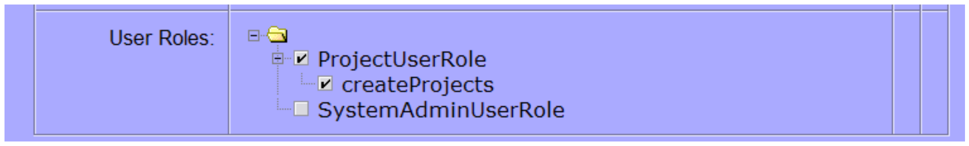
The user name must be unique. If the chosen name is already in use by another user, the system reports an error message to the right of the username field as shown in Figure 92.



Username: project The username conflicts with an existing User

Figure 92: Username Already in Use

The user roles and permissions are used to authorize the user access to different features of the system. The "Project User Role" grants the user access to the project features of the system. The "Create Projects" permission under the project user role grants the user the ability to create new projects. The "System Admin User Role" grants the user access to the user administration function of the system. Figure 93 shows an example of a user with the project user role and permission to create new projects.



User Roles:
- ✓ ProjectUserRole
  - ✓ createProjects
- ☐ SystemAdminUserRole

Figure 93: User Roles and Permissions

Roles and permissions can be changed after a user is created, but a user must always have at least one role. To disable a user's access to the system the password should be reset.

Chapter 5 Summary and Conclusions

This chapter summarizes the successes and failures of the project, directions of future work, and the conclusions of the original premise.

Lessons Learned

The key lessons learned are that the scope of the project was way too big and too many unfamiliar technologies were used. However, I learned a lot about engineering a large scale project, the benefits of the command processor pattern, and the intricacies of a lot of new technologies.

Identifying the use cases ahead of time and then using an iterative process to implement them helped gauge the effort needed to complete the project and allow for adjustments as the project progressed. Implementing to use cases also allowed usable pieces of the system to be built incrementally. When use cases were dropped no effort was wasted because the work done in each iteration stayed focused on the use cases being implemented. The fact that there were over seventy use cases in the proposal should have been taken as a sign that the scope was too big. It probably would have been better to set a limit on the use cases before starting the project to keep the scope manageable.

Natural language processing is hard. There are a lot of open source tools and databases, but many are standalone tools and not easy to integrate.

Conclusion

The original premise of the project was that writing high quality requirements is difficult and a tool that supports collaboration among the stakeholders and automated assistance would make it easier. Unfortunately I never got to test the theory with a team of stakeholders so I don’t know if the discussion feature will improve collaboration.

Due to the difficulties with natural language processing the scope of automated assistance ended up being small. Identifying glossary terms and being able to replace duplicate terms for the same concept in the requirements text are probably the most useful.

Future Work

This section touches on aspects of Requel that could be improved with some ideas on things to try.

Natural Language Processing

The natural language processing component of the application needs a lot of work. In particular the word sense disambiguation component works poorly. While the semantic similarity function works fairly well, it only disambiguates words of the same part of speech and does much better at nouns than other parts of speech. The co-reference disambiguation using SemCor sentences and WordNet glosses was totally useless on actual requirements data. The only way this method seems like it would work is to use a large domain specific corpus

The Stanford parser worked fairly well with the bundled grammars, but it would probably work better trained on requirements specifications.

The semantic role labeler performed fairly well on test data with simple sentences and hand coded word senses, but performed poorly on actual requirements. This was partially due to the poor performance of the word sense disambiguation, but mainly due to the limited verb coverage and syntax to semantic role mappings in VerbNet.

The Stanford named entity recognizer didn’t perform too well, again training on domain specific data would probably improve its performance.

Domain Knowledge

The original scope of the project included a domain component with knowledge about concepts relevant to an application domain. At a minimum adding a domain lexicon of words and senses could improve the natural language processing performance.

On the requirements engineering side, having a domain specific glossary and example requirement elements could help jump start projects.

User Interface

The user interface works ok, but required a huge up front effort to create the UI framework on top of Echo2 to manage navigation and panels. Some of the Echo2 components are a bit rough, such as the tab set. Panels with long titles take up the whole tab section, and the tabs extend off the screen if a bunch of panels are opened. Writing new components in Echo2 that aren’t just composites of existing components is difficult and requires a mix of Java and Javascript.

It would be interesting to try a framework like Flash or JavaFX to see if it would be more productive.

References

Adolph, S., Cockburn, A., and Bramble, P. (2002) Patterns for Effective Use Cases. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Alexander, I. (2007) Requirements Tools Web page Available at http://easyweb.easynet.co.uk/~iany/other/vendors.htm

Banerjee, S. (2002) Adapting the Lesk Algorithm for Word Sense Disambiguation to WordNet (Masters Dissertation, University of Minnesota, 2005) Retrieved June 4, 2008 from http://www.d.umn.edu/~tpederse/Pubs/banerjee.pdf

Brooks, F. (1987) No Silver Bullet Essence and Accidents of Software Engineering. Computer 20, 4 (Apr. 1987), 10-19.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996) Pattern-oriented software architecture: a system of patterns. New York: John Wiley & Sons, Inc.

Ciemniewska, A., Jurkiewicz, J., Olek, Ł., and Nawrocki, J. (2007) Supporting Use-Case Reviews. Business Information Systems vol. 4439/2007. Springer Berlin: Springer-Verlag. Draft Retrieved October 1, 2007 from http://www.cs.put.poznan.pl/lolek/homepage/Research_files/07-BIS.pdf

Christel, M. G. and Kang, K. C. (1992) Issues in Requirements Elicitation. Technical Report (CMU/SEI-92-TR-12) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University Retrieved July 9, 2007 from http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr12.92.pdf

Cockburn, A. (2000). Writing Effective Use Cases. Boston, MA: Addison-Wesley.

Conklin, J. and Begeman, M. L. (1988). gIBIS: a hypertext tool for exploratory policy discussion. ACM Trans. Inf. Syst. Vol. 6, issue 4 pp. 303-331. Retrieved September 12, 2007 from http://www.cs.hut.fi/Opinnot/T-93.850/2005/Papers/gIBIS1988-conklin.pdf

Dražan, J. and Mencl, V. (2007). Improved Processing of Textual Use Cases: Deriving Behavior Specifications. In Proceedings of the 33rd Conference on Current Trends in theory and Practice of Computer Science. J. Leeuwen, G. F. Italiano, W. Hoek, C. Meinel, H. Sack, and F. Plášil (Eds.) Lecture Notes In Computer Science, vol. 4362.

Berlin: Springer-Verlag. Retrieved February 2, 2008 from http://dsrg.mff.cuni.cz/publications/DrazanMencl-ImprovedUC-SOFSEM2007.pdf

Eberlein, A. (1997) Requirements Acquisition and Specification for Telecommunication Services. PhD thesis, University of Wales, Swansea, UK. Retrieved August 3, 2007 from http://www2.enel.ucalgary.ca/People/eberlein/publications/PhD_Thesis.pdf

Fellbaum, C. (Ed.). (1998) WordNet: An Electronic Lexical Database, Cambridge, MA: MIT Press.

Finkelstein, A. (1994) Requirements Engineering: a review and research agenda. In Proceedings of the First Asia Pacific Conference on Software Engineering, New York, NY: IEEE CS Press, pages 10-19.

Fowler, M. (2002) Patterns of Enterprise Application Architecture. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Fowler, M. (2004) Inversion of Control Containers and the Dependency Injection pattern. Retrieved February 8, 2008 from http://martinfowler.com/articles/injection.html

Gale, W. A., Church, K. W., and Yarowsky, D. (1992) One sense per discourse. In Proceedings of the Workshop on Speech and Natural Language. Morristown, NJ: Association for Computational Linguistics. Retrieved March 6, 2009 from http://acl.ldc.upenn.edu/H/H92/H92-1045.pdf

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Goncalves, A. (March 2007) Generate an XML Document from an Object Model with JAXB 2. DevX website article Retrieved January 22, 2008 from http://www.devx.com/Java/Article/34069

Johnson, R., Hoeller, J., Arendsen, A., Sampaleanu, C., Harrop, R., Risberg, T., et al. (2007.) The Spring Framework - Reference Documentation Version 2.5. Retrieved February 8, 2008 from http://static.springframework.org/spring/docs/2.5.x/spring-reference.pdf

Jurafsky, D., and Martin, J. (2008). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Upper Saddle River, NJ: Prentice-Hall.

Kipper-Schuler, K. (2005) VerbNet: A broad-coverage, comprehensive verb lexicon. (Doctoral Dissertation, University of Pennsylvania, 2005) Retrieved March 6, 2008 from http://verbs.colorado.edu/~kipper/Papers/dissertation.pdf

Klein, D., and Manning, C. (2003) Fast Exact Inference with a Factored Model for Natural Language Parsing. In Advances in Neural Information Processing Systems 15 (NIPS 2002), Cambridge, MA: MIT Press, pp. 3-10. Retrieved January 30, 2007 from http://www-nlp.stanford.edu/~manning/papers/lex-parser.pdf

Lesk, M. (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In V. DeBuys, (ed.) Proceedings of the 5th Annual international Conference on Systems Documentation. SIGDOC '86. New York: ACM Press, pp. 24-26.

Mencl, V. (2004) Deriving Behavior Specifications from Textual Use Cases, in Proceedings of Workshop on Intelligent Technologies for Software Engineering. Linz, Austria: Oesterreichische Computer Gesellschaft. pp. 331-341. Retrieved January 23, 2007 from http://dsrg.mff.cuni.cz/publications/Mencl-DerivingBehSpec-WITSE04.pdf

Mihalcea, R. (1998) SEMCOR - Semantically tagged corpus. Retrieved February 4, 2008 from http://www.seas.smu.edu/~sanda/research/semcor.ps.gz

Mylopoulos, J. and Castro, J. (2000) Tropos: A Framework for Requirements-Driven Software Development. In Brinkkemper, J. and Solvberg, A. (eds.), Information Systems Engineering: State of the Art and Research Themes. Lecture Notes in Computer Science, Springer-Verlag. Retrieved August 22, 2007 from http://www.troposproject.org/papers_files/Fossil.pdf

Nawrocki J., Olek £. (2005) Use-cases engineering with UC Workbench, in Zieliñski, K. and Szmuc, T. (eds.), Software Engineering: Evolution and Emerging Technologies, volume 130, pages 319–329. Amsterdam: IOS Press. Retrieved October 1, 2007 from http://www.cs.put.poznan.pl/lolek/homepage/Research_files/UCWorkbench.pdf

Nuseibeh, B. and Easterbrook, S. (2000) Requirements Engineering: A Roadmap. In A. Finkelstein (ed.) The Future of Software Engineering (pp. 35-46). (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE'00). Los Alamitos, CA: IEEE Computer Society Press. Retrieved July 14, 2007 from http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

Ort, E. and Mehta, B. (March 2003) Java Architecture for XML Binding (JAXB) from Sun Developer Network website Retrieved January 22, 2008 from http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

Potts, C., Takahashi, K., and Antón, A. (1994) Inquiry-Based Requirements Analysis. IEEE Software vol. 11-2 pp. 21-32. Retrieved July 14, 2007 from http://www4.ncsu.edu/~aianton/pubs/ieeeSW.pdf

Reubenstein, H. B. and Waters, R. C. (1991) The Requirements Apprentice: Automated Assistance for Requirements Acquisition. IEEE Transactions on Software Engineering vol. 17 no. 3, pp. 226-240

Rich, C., and Waters, R. (1986). Toward a Requirements Apprentice: On the Boundary between Informal and Formal Specifications. (AI Memo AIM-907). Cambridge, MA: MIT, Computer Science and Artificial Intelligence Lab. Retrieved April 30, 2007 from http://hdl.handle.net/1721.1/5516.

Rich, C., and Waters, R. (1987). The Programmer's Apprentice Project: A Research Overview. (AI Memo AIM-1001). Cambridge, MA: MIT, Computer Science and Artificial Intelligence Lab. Retrieved June 15, 2007 from http://hdl.handle.net/1721.1/6054.

Rich, C., and Waters, R. (1990). The Programmer's Apprentice. New York, NY: ACM Press.

Robinson, W. N. and Volkov, S. (1997) A meta-model for restructuring stakeholder requirements. In Proceedings of the 19th international Conference on Software Engineering. ICSE '97. ACM Press, New York, NY, 140-149. Retrieved August 8, 2007 from http://www.cis.gsu.edu/~wrobinso/papers/icse97.ps

Ruppenhofer, J., Ellsworth, M., Petruck, M., Johnson, C., and Scheffczyk, J. (2006) FrameNet II: Extended Theory and Practice Retrieved January 25, 2008 from http://framenet.icsi.berkeley.edu/index.php?option=com_wrapper&Itemid=126

Schuler, K. 2005 VerbNet: a Broad-Coverage, Comprehensive Verb Lexicon. Doctoral Thesis. University of Pennsylvania. Retrieved January 25, 2008 from http://verbs.colorado.edu/~kipper/Papers/dissertation.pdf

Seco, N., Veale T., and Hayes, J. (2004) An Intrinsic Information Content Metric for Semantic Similarity in WordNet. Technical Report. Dublin, Ireland: University College Dublin Retrieved October 8, 2008 from http://afflatus.ucd.ie/Papers/ecai2004b.pdf

Sharp, H., Finkelstein, A., and Galal, G. (1999) Stakeholder Identification in the Requirements Engineering Process. In Proceedings of the 10th international Workshop on Database & Expert Systems Applications, IEEE Computer Society, Washington, DC. Draft Retrieved September 5, 2007 from http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/stake.pdf

Simpson, T., and Dao, T. (2005, Oct 1). WordNet-based semantic similarity measurement. Retrieved April 4, 2008 from http://www.codeproject.com/KB/string/semanticsimilaritywordnet.aspx

Smith J. D. (1997) EColabor: Collaborative Elaboration of Documents. NTT Multimedia Communications Laboratories. Berkeley Multimedia and Graphics Seminar. Retrieved July 3, 2007 from http://bmrc.berkeley.edu/courseware/cs298/spring97/w5/slides.ppt

Takahashi, K., Potts, C., Kumar, V., Ota, K., and Smith, J. D. (1996) Hypermedia Support for Collaboration in Requirements Analysis. In Proceedings of the 2nd international Conference on Requirements Engineering (ICRE '96) (April 15 - 18, 1996). ICRE. IEEE Computer Society, Washington, DC, 31.

van Lamsweerdem, A. (2001) Goal-Oriented Requirements Engineering: A Guided Tour Retrieved July 15, 2007 from http://www.info.ucl.ac.be/Research/Publication/2001/RE01.pdf

Weigers, K. E. (1999). Software Requirements. Redmond, WA: Microsoft Press.

Yu, E. S. (1997) Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In Proceedings of the 3rd IEEE international Symposium on Requirements Engineering (Re'97) (January 05 - 08, 1997). RE. IEEE Computer Society, Washington, DC, 226. Draft Retrieved August 21, 2007 from http://www.cs.toronto.edu/pub/eric/RE97.pdf

Appendix 1 Application Code

Note the use of the 'Appendix' style. You can have as many appendices as you need. For instance, you might have an extended sample of input/output in its own Appendix. Use the style 'Appendix' for each appendix title.

Organize the Application Code appendix by code type (Java, JSP, HTML, XML, etc), with each code type having a second-level heading.   The name of each file should be a third-level heading.

Code should be single-spaced, Courier New, 10 Point font. Use the style ‘code-page’ to achieve this layout. Except for the first code type in the Appendix, each code type should start a new page, and except for the first file listed in a given code type, each file should start a new page.

Be sure to have a paragraph here describing what kinds of files you have and how you’ve chosen to organize them.

Java Code

Code in the Appendix is flush with the left margin, not indented

Sample.java

```
/**
Sample code
 **/
class Sample implements Serializable {

}
```

HTML Files

Sample.html