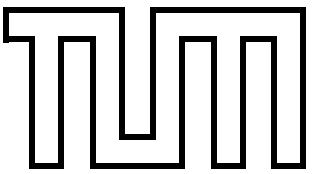


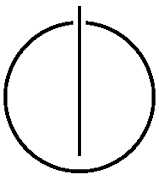
FAKULTÄT FÜR INFORMATIK

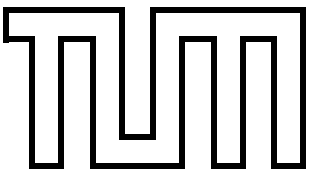
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

A Web-Based Front End to Scenario-Based Procurement with Quantity Rebates

Philipp Reichart





FAKULTÄT FÜR INFORMATIK

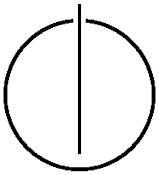
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

A Web-Based Front End to Scenario-Based Procurement with Quantity Rebates

Eine Web-Basierte Schnittstelle zu Szenario-Basierter Beschaffung mit Mengenrabatten

Author:	Philipp Reichart
Supervisor:	Prof. Dr. Martin Bichler
Advisor:	Dipl.-Inf. Stefan Schneider
Date:	13. November 2009



I assure the single handed composition of this bachelor's thesis only supported by declared resources,

München, 13. November 2009 Philipp Reichart

Abstract

Diese Arbeit beschreibt die Umsetzung einer web-basierten Schnittstelle zum SPQR-System mit Fokus auf Benutzbarbkeit, Einfachheit, Wartbarkeit und Best Practices bei zeitgemäßer Entwicklung von Java-Webanwendungen. Die grundlegenden Technologien zur Erstellung web-basierter Anwendungen in der Programmiersprache Java werden erklärt und die Wichtigkeit der Verwendung von Frameworks zur Entwicklung eleganter, erweiterbarer und wartbarer Lösungen dargestellt. Verschiedene Metriken erleichtern und begründen die Auswahl des geeignetsten Frameworks aus einer Vielzahl von Kandidaten. Es folgt die Beschreibung des letztendlich verwendeten Frameworks sowie des Umsetzungsprozess mit besonderem Augenmerk auf die nutzenstiftende und intuitiv verständliche Darstellung von Schlüsselkomponenten.

Abstract

This thesis describes the effort to implement a web-based front end to the SPQR system with focus on ease of use, simplicity, maintainability and best practices in current Java web application development. The fundamental technologies used in web-based applications in the Java Programming Language are introduced along side the importance of using a web framework for developing elegant, extensible and maintainable solutions. Various metrics for selecting the best fit framework from a multitude of candidates are presented. The final choice and implementation process is described with special attention paid to providing added value by intuitively visualizing key components.

Contents

ASF Apache Software Foundation

CSS Cascading Stylesheets

DRY Don't Repeat Yourself

EL Expression Language

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

I18N Internationalization

IBIS Internet-based Information Systems

IDE Integrated Development Environment

IRC Internet Relay Chat

JCP Java Community Process

JEE Java Enterprise Edition

JRE Java Runtime Environment

JSF Java Server Faces

JSP Java Server Page

JSR Java Specification Request

JSTL Java Standard Tag Library

L12N Localization

MVC Model View Controller

OGNL Object Graph Notation Language

POJO Plain Old Java Object

RMI Remote Method Invocation

SPQR Scenario-Based Procurement with Quantity Rebates

UI user interface

URL Uniform Resource Locator

W3C World Wide Web Consortium

WAR Web Application Archive

1 Problem Statement

The department of Internet-based Information Systems (IBIS) at TU München analyzes and models business decisions to solve and optimize complex decision problems [**?**]. Scenario-Based Procurement with Quantity Rebates (SPQR) is one of the tools used for the purpose of calculating the optimal combination of offers in procurement auctions [**?**].

The inputs to SPQR are a scenario containing a given setup, business rules set forth by the auctioneer and submissions coming from bidding parties. A setup contains a given number of suppliers, items and their demand. Business rules may enforce lower and upper bounds on the quantity and money spend on suppliers and items. Submissions consist of offers for an item as well as optional payment modifiers and discount conditions both bound to sets of items offered by a single supplier. A payment modifier is able to shift prices both ways when its discount conditions are met, i.e. it can act as both markup and rebate. Discount conditions may exclude other discounts from applying.

The output – called result – features the overall amount of money spend, the number of active discounts and rebates as well as the amount saved by each. For each winning supplier, the amount spend and quantity purchased and for each item contained in a winning offer, the amount spent is listed.

In its current form, SPQR runs as a command-line application accepting and outputting custom formatted plain-text files which are barely human-readable, hard to understand quickly and tedious to handle. This thesis provides an easy-to-use and intuitive web-based front end to the existing command-line application. Key elements of the data model are visualized in a tabular and graphic manner to aid quick understanding.

2 Analysis

2.1 Requirements

The web-based front end should be implemented in the Java Programming Language like the existing application code. This reduces friction between the back and front end by allowing to reuse the existing model and solver classes in the web-based front end.

Changes to the existing data model should be avoided so development of the back end can continue during the course of the thesis. This should also lessen the propability of introducing changes incompatible with the native solver libraries which are not the focus of the thesis.

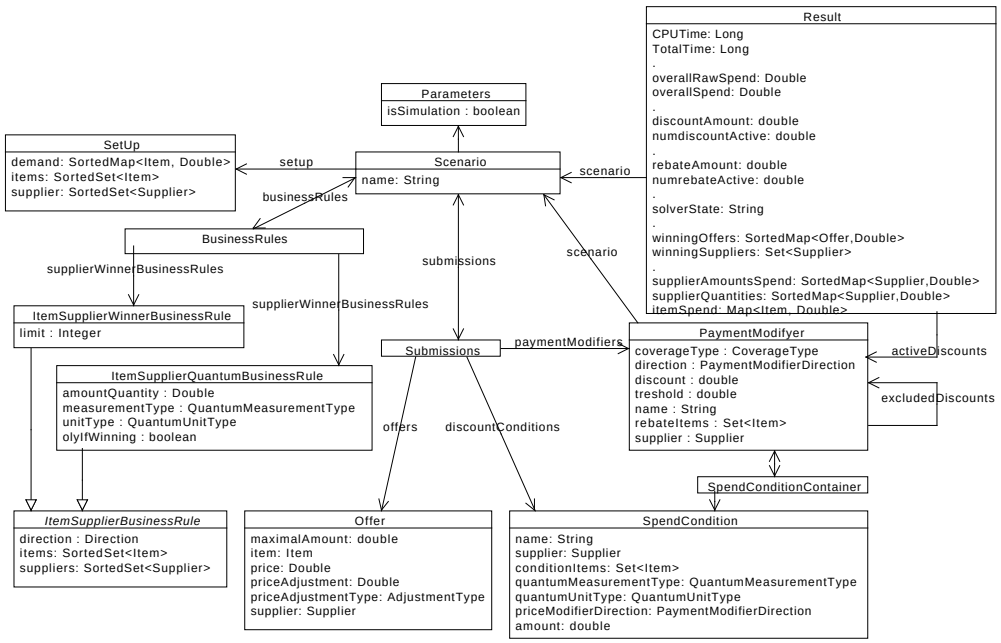


Figure 1: The data model of SPQR. Custom types (fig. 2) and references to the `Item` and `Supplier` (fig. 3) classes are not displayed in the diagram for clarity. References shown in the diagram are not listed as attributes of the class itself.

The web-based front end should allow for easy future extension and changes to be made without requiring specialist knowledge or a long learning phase for future developers. This will be facilitated by choosing a mature web application framework and using it consistently throughout the development effort.

Key elements of scenario and result should be visualized intuitively to aid in the quick understanding of the data model and contents.

2.2 Class Diagram of Data Model

The data model consists of various classes containing only accessors and mutators for the properties. The actual business logic of solving the optimization problem is performed by a separate solver library taking an object tree built after the model in figure 1 as input. The central entity is the `Scenario` class from which all other entities can be reached.

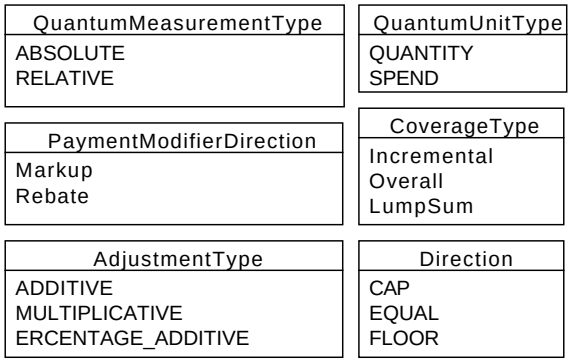


Figure 2: Custom types implemented as `Enum` in the data model.

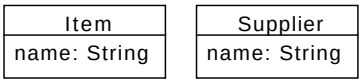


Figure 3: The `Item` and `Supplier` types used in the data model.

2.3 Main Functionality

The main functionality of the web-based front end to SPQR can be categorized into four parts:

- Load/Randomize Scenario and Create Setup

  These three actions represent means to provide input to the SPQR system either from existing text-based scenario files, a parametrised random generator or manually creating the set up.

- Review Scenario

  The scenario is displayed in a structured way with key elements visualized to allow for a quick understanding of the items, suppliers, business rules and submissions contained in a scenario.

- Solve Scenario

  Invokes the native solver library in the back end and informs the user of solving progress.

- View Result

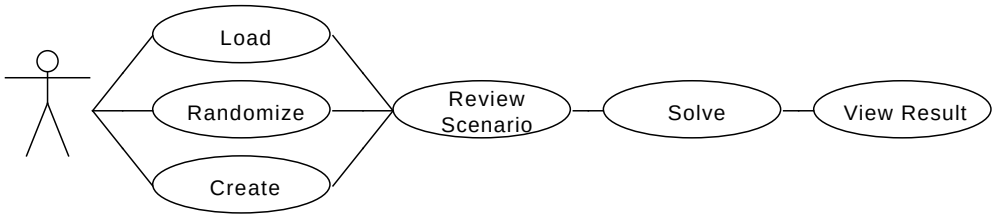


Figure 4: The usual workflow from input to result.

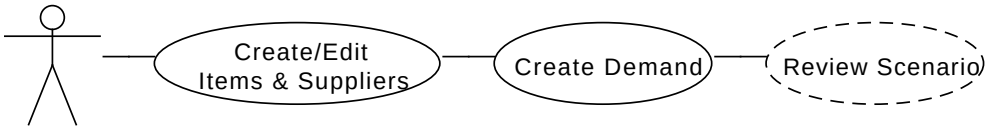


Figure 5: The workflow of manually creating or editing a set up, finally leading to the scenario review as shown in figure 4.

The result of finding an optimal solution for the scenario is displayed, again visualizing important aspects to help in intuitively grasping the outcome.

2.4 Use Cases and Workflows

The usual workflow as seen in figure 4 always starts with providing a Scenario as input to the web-based front end. After reviewing the scenario contents, the user invokes the solver and will be presented with the result.

The use case of creating the set up manually shown in figure 5 can be further refined into creating, or editing for existing set ups, the items and suppliers and then specifying the demand per item.

2.5 Visualization of Key Elements

Key elements of the set up, scenario and result model classes are visually enhanced as tables or charts to allow for easier understanding.

Tabular display can be used for large collections of model classes which have uniform fields like the business rules, submissions, and payment modifiers with their spend conditions.

Candidates for charts are maps of model classes to numeric values like demand, and the various "spend per entity" or "amount per entity" attributes of the result class.

3 Technologies in Web-Based Applications

3.1 Fundamentals of web-based applications

Web-based applications work in a fundamentally different way than regular deskop-based software. Whereas in the latter, the developer has complete control over the sequence of actions, the overall flow of control and data in a web-based application is defined by the request-response scheme set forth by the Hypertext Transfer Protocol (HTTP): A client sends a request to a web server which in turn invokes application code and generates a response to be sent back to the client. The client usually takes the form of a web browser but may as well be a non-graphical program or even another web-based application. Both request and response usually consist of a header containing control instructions and body with the actual data payload.

Listing 1: HTTP headers for a sample GET request and response.

```
1  GET / HTTP/1.1
2  Keep-Alive: 300
3  Connection: keep-alive
4
5  HTTP/1.x 200 OK
6  Date: Mon, 26 Oct 2009 12:16:18 GMT
7  Server: Apache
8  Last-Modified: Thu, 23 Oct 2008 20:45:22 GMT
9  Content-Length: 213
10 Content-Type: text/html
```

A Uniform Resource Locator (URL) is used to address specific resources on a web server and allows additional parameters to be specified for a request on which the receiving application code can act on. Different methods can be invoked on a URL, the most common ones used in web-based applications being GET and POST.

Listing 2: The structure of a URL

```
1  http://example.com/login?user=scott&password=tiger
```

The former submits all request parameters appended to the URL, while POST delivers parameters in the request body allowing for larger payloads

and more confidential transmission as the most clients only display parameters appended to the URL but not those in the request body. Other methods include HEAD to only receive the header but not body of a resource, DELETE and PUT to remove or create a resource on the server and the little supported TRACE to get a detailed explanation on how a request got executed by the server.

As a further constraint, HTTP does not persist state, so a web-based application either has to manage its own state or all state information has to be transferred on every request, e.g. by parameters. Most current web-based applications do not transfer the actual state on every request but only a lookup key called *session ID* to a server-side repository. This session ID is either added as request parameter into every URL (called "rewriting") or implemented as a 'Cookie', a special kind of HTTP header that is sent back and forth by server and client on every request.

Whereas desktop-based applications usually define their user interface (UI) directly by means of a toolkit providing common UI components and controls, web-based applications provide a textual description of their UI with every request in Hypertext Markup Language (HTML) which is interpreted and displayed by a web browser. To navigate between pages, the user clicks on hyperlinks or uses forms to submit information to the server. Although the parsing and rendering of the various versions of HTML is formally specified by the World Wide Web Consortium (W3C), compatibility and visual fidelity has been a problem between major web browsers up until recently. The recent and upcoming versions of XHTML and HTML 5 more rigidly specify the often misunderstood details.

Listing 3: A simple HTML document with a short greeting.

```
1 <html>
2 <head>
3 <title>Welcome to example.com</title>
4 </head>
5 <body>
6 <p>Hello visitor!</p>
7 </body>
```

In the beginning, HTML mixed both the structure and visual presentation of a document, which made it difficult to deliver the same document to different browsers and different devices (screen, printer, mobile). To remedy this and more cleanly separate structure from presentation, Cascading Stylesheets (CSS) where introduced in 1996. Initially also supported inconsistently across browsers, CSS is now considered the default layout language for HTML documents.

Listing 4: A CSS stylesheet setting all paragraphs red and in sans-serif font.

```
1 p {
2   font-family: sans-serif;
3   color: #F00;
4 }
```

To allow for more dynamic user interaction with web-based applications consisting mainly of static HTML documents, JavaScript was introduced to enable a web-based application to directly modify the structure and presentation of a document in the user's web browsers without a mandatory roundtrip to the web server. Recent developments try to integrate JavaScript unobstrusively into web pages to allow web-based applications to fail gracefully when no JavaScript is present (e.g. on mobile devices) and to achieve separation of behavior (JavaScript) from document structure (HTML) and presentation (CSS).

Listing 5: A JavaScript function to display a message on the click of a button.

```
1  <html>
2  <head>
3  ...
4  <script type="text/javascript">
5  function popup() {
6    alert("Your lucky number is " + 100*Math.random());
7  }
8  </script>
9  </head>
10 <body>
11   <button onclick="popup()">Lucky Number</button>
12 </body>
13 </html>
```

3.2 Web-based applications in Java

In Java, web-based applications are usually built upon the Servlet API, first released publicly in November 1998. Servlets are special Java classes executed by a web container which abstracts most of the technical details of HTTP and provides an object-orientied interface it and allows the Servlet programmer to focus more on the actual business logic.

A Servlet follows a defined life cycle of initilization, the handling of one or more requests and finally destruction. Various utility methods and classes to access and manipulate HTTP headers and parameters are provided as well as

automatic session handling through cookies or URL rewriting. Data can be temporarily persisted in various scopes (page, request, session, or application) and shared among all code involved in handling a request. Persistent storage is not provided by the Servlet API.

Listing 6: A basic Servlet greeting the user (without import statements).

```
1 public class MyServlet extends HttpServlet {
2   protected void doGet(HttpServletRequest req,
3     HttpServletResponse resp)
4       throws ServletException, IOException {
5     resp.getWriter().println("Hello visitor!");
6   }
7 }
```

Because of the cumbersome generation of content-centric documents with a Servlet which is mainly code and requires explicit print-statements for each line of HTML to be outputted, the Servlet concept was complemented by that of the JSP. The contents of a JSP are mainly content outputted by default and code has to be defined explicitly. This allowed for a more natural separation of code-based business logic into Servlets and HTML-based view logic into JSPs.

Listing 7: A basic JSP giving the current date.

```
1  <%@ page import="java.util.Date" %>
2  <%@ page import="java.text.SimpleDateFormat" %>
3  <html>
4  ...
5  <body>
6  <%
7  DateFormat df = new SimpleDateFormat("dd.MM.yyyy");
8  %>
9  <p>It's now <%= df.format(new Date()) %></p>
10 </body>
11 </html>
```

As allowing code in content-centric JSPs proofed to be complimentary to the problem of Servlets containing too much HTML, the concept of tag libraries was introduced. A tag library fits nicely with the structured contents of an HTML document but is interpreted by the server and may generate additional HTML. Implemented as Java code, tag libraries integrate with the Servlet and JSP APIs to allow the same level of functionality as code directly written into JSPs but with far greater separation and potential for reuse.

Listing 8: A basic JSP giving the current date with a custom tag library.

```
<%@ taglib uri="/WEB-INF/mytags.tld" prefix="my"
<html>
...
<body>
<p>It's now <my:currentDate/></p>
</body>
</html>
```

Listing 9: A the tag handler for the "currentDate" tag used in figure 8.

```
...

public CurrentDateTag extends TagSupport {

  DateFormat df = new SimpleDateFormat("dd.MM.yyyy");

  public int doStartTag() throws JspException {
    try {
      pageContext.getOut().print(df.format(new Date()));
    } catch (Exception ex) {
      throw new JspTagException(ex);
    }
      return SKIP_BODY;
  }

  public int doEndTag() {
    return EVAL_PAGE;
  }
}
```

Tag libraries proved to allow much cleaner programming of web-based application in Java but were also rather complex to set up, requiring both code and deployment descriptor. To further ease development and not revert back to writing full-fledged code into JSPs, in 2002 the Java Standard Tag Library (JSTL) was introduced to take care of the common tasks of control flow, general view logic, locale-based formatting for purposes of I18N and L12N. In 2006, the JSP specification was amended to support a unified Expression Language (EL) to simplify the most common tasks even more while fully integrating into the existent technology stack.

Listing 10: A basic JSP giving the current date with a custom tag library.

```
<html>
```

```
2 ...
3 <body>
4 <jsp:useBean id="now" class="java.util.Date" />
5 <p>It's now <fmt:formatDate value="${now}"
6   pattern="dd.MM.yyyy" /></p>
7 </body>
8 </html>
```

3.3 Web Frameworks

Most web-based applications share the overall structure and control flow of parsing and validating request parameters, invoking business logic and then rendering the result by forwarding request handling to a separate view. To relieve developers from writing common boilerplate code and control structures again for every web-based application, frameworks were developed based on the rather low-level Servlet and JSP APIs. Frameworks usually incentivise the developer to submit to a more stringent programming model with a focus on separation of code and HTML for improved reusability by commonly providing support for

- Separation of concerns through design patterns like Model View Controller (MVC). The framework usually provides at least part of the controller role including mapping requests to their respective controller classes and providing overall page flow,
- Automated mapping of request parameters to model classes and vice versa, e.g. "binding" model properties to UI elements. This allows the developer to focus on implementing business logic instead of handling HTTP details both when taking input from requests and when outputting data,
- Declarative and programmatic validation of input, including communication of validation failures to the end user, e.g. by highlighting an invalid form field and displaying an error message next to it,
- Internationalization (I18N) and Localization (L12N), i.e. displaying UI labels and messages in different languages and formatting numbers, currencies and dates according to the preferred locale of the end user,
- Tag libraries or custom template mechanisms to keep the view components free from logic and provide additional functionality that would

require lots of effort to program manually, e.g. tabular sortable display of data or automated generation of input/edit forms from model classes.

Most web frameworks can be classified as being either request/response-based, staying close to the fundamental concept of HTTP, or component-based, abstracting the flow of web-based applications to a high level similar to desktop-based applications where business logic is triggered by specific events coming from form submissions or button presses.

3.4 Metrics for choosing a framework

For the scope of this thesis, only Open Source frameworks were considered as possible candidates. Aside from cost, most of the following selection criteria and goals cannot be easily applied to closed-source software as their development process does usually not provide the necessary level of transparency.

To choose from the many available web application frameworks for the Java Programming Language, goals and criteria have to be defined and a rationale and relevance specified for each to provide for objective selection of the best fit framework for the development effort at hand.

The following goals describe abstract attributes and qualities pertaining to the overall handling of a framework over the application lifetime, each divided into multiple concrete criteria to evaluate several web frameworks:

3.4.1 Ease of learning

A good and elegant framework helps very little if the barrier to entry is so high it drives away most aspiring developers. Many frameworks provide only source code and binaries as the bare minimum and little to no documentation geared towards new developers. This lack of entry-level documentation not only prevents new developers from learning a framework but also puts a large burden and source of frustation on maintenance developers that do not have the choice but to learn it.

Is a quick start guide and/or tutorial available? Many frameworks supply an API reference and full-fledged manual but fail to provide an easy entry for developers new to the framework. A short guide or step-by-step instructions are much more pleasant to experience a new framework than reading hundreds of manual pages front to back.

Documentation and Community Support After the first steps, the most commonly referred to documentation for most frameworks will be the

API reference to learn about the classes a framework has to offer and "How To's" that describe common problems and their solution as intended by framework, e.g. how to upload a file and make it accessible to business logic. For more specific or timely matters, a forum, mailing list or Internet Relay Chat (IRC) channel provide more appropriate platforms. A framework with a strong and alive community is preferable, as it will be more likely to provide help with specific problems and provide workarounds to open bugs. Community Support is often not limited to the framework's website itself, but may also be found on large programming-related websites like Experts Exchange [**?**] or Stack Overflow [**?**].

Standards Support Does the framework support official and/or de-facto standards or does it reinvent the wheel, forcing a developer to learn new ways to common tasks that have long been solved by the programming community at large? For the Java Programming Language, official standards are those set forth by the Java Community Process (JCP) and defined in a Java Specification Request (JSR). De-facto standards are usually mature projects developed on large Java-based ecosystem providers like the Apache Software Foundation (ASF) with their Jakarta meta-project for Java components.

3.4.2 Ease of development

Effortless Programming A major part of the development of a web-based application is spent implementing the requirements with the functions provided by the web framework of choice, so a framework should provide an intuitive and elegant programming model. Principles like Don't Repeat Yourself (DRY) and "convention over configuration" should be followed to minimize repetition in code and configuration files and prevent resulting errors due to subtle typing and naming mistakes, e.g. mixing the case of a class name in the class declaration and deployment descriptor.

The framework should simplify common tasks as much as possible while still allowing one-off requirements to be implemented outside the frameworks boundaries, e.g. by providing access to the underlying Servlet/JSP objects if required.

Unique Features and Restrictions What unique features and restrictions differentiate a framework from its competitors and what is the intended optimal environment to use the framework in? Also of interested and

a sign for a well-thought out framework is an explicit description of what the framework is *not* meant to do, i.e. where it deliberately fails to assist the developer.

Testability Unit testing of code has become a best practice to guarantee the intended behavior and quality of code. The less dependencies a web framework imposes on the code written for a task, the easier this code is to test. Some frameworks go out of their way to even remove dependencies on the framework itself from application code.

Tool Support Depending on the complexity of the technology and the development task at hand, tool support is an important factor in speeding up the implementation and decrease the propability of errors and manual creation of boiler plate code. Preferrably, a framework should provide a plug-in for the developer's Integrated Development Environment (IDE) of choice.

3.4.3 Maturity and Future Proofness

Maturity The age of the code base, number of contributors and releases as well as the existence and active use of bug tracking is usually a good sign of a mature project. These are more likely to deliver higher quality software with less bugs and an overall better development experience due to the time and effort already spent on fixing or at least documenting shortcomings in earlier releases.

Release Management Any software component considered to be used in a development project should be screened carefully for it's release management to avoid surprises when upgrading between versions. A good release management process delivers pre-releases of upcoming new versions to test for compatiblity early and provides detailed upgrade and deprecation guides as well as lists of recently fixed and still known open issues.

Production Use A good indicator for the stability and usability of a framework is its use by large companies or institutions. Sometimes, large users will provide valuable insights into their use of and possibly migration to the respective framework, allowing to evaluate a framework against real world requirements.

Active Development The web framework should be under active development with a live community to guarantee continuation of releases, bug fixes and support in the future. Switching frameworks is possibly the

worst mishap in terms of development time and will most likely proove difficult when unique features offered by the old framework have to be migrated. Unless willing and able to support and bugfix the framework itself, dead or dying frameworks should be avoided.

3.5 Candidate Frameworks

3.5.1 Plain Servlets and JSPs

Every project to implement a web-based application should – even if only briefly – consider using plain Servlets and JSPs to fulfil the requirements. Although for any non-trivial project, this will quickly lead to the realization of being prohibitively time and resource consuming, it provides insights into what exactly are the project-specific requirements a web application framework has to meet. The most common or most complex functionality required should be the one assisted in most by the framework of choice.

3.5.2 Struts 1

Founded in 2004 on the ASF's Jakarta project, Struts [**?**] was one of the first widely used web application frameworks and represents the ancestor of most other request/response-based web frameworks built around the MVC pattern. It solves many of the most pressing issues with the plain Servlet/JSP approach by roughly providing all of the functionality listed in section 3.3. Drawbacks are a very tight coupling of application code with Struts classes, lots of boilerplate code in violation of DRY and large amounts of configuration split over multiple files. The common way for Struts 1 was to write an ActionForm representing an HTML with all its fields, an ActionBean for handling the application logic and one or more JSPs as views to forward to after the logic has executed – each of them configured with a few lines of XML. Although considered one of the most successful web frameworks for the Java Programming Language, the shortcomings of Struts 1 spawned a myriad other frameworks to improve upon it.

3.5.3 Stripes

The Stripes Framework [**?**] was created to overcome the "lot of small things that really add up" and decrease productivity when developing with Struts 1.

It's main selling points were the reduction of artifacts for request processing (only an ActionBean and JSPs), a leaner view (less taglibs, more

plain HTML for rapid prototyping of the UI) and full support for deep property binding, i.e. a view can read/write `scenario.setup.supplier` and the framework will automatically follow the object graph and create non-existing elements on the fly [?]. Struts 1 would fail when any part of the object graph was `null`, resulting in the need for a lot of repetitive code.

3.5.4 Spring MVC

Spring was traditionally a framework to structure the business logic living in the middle tier of an application but has in recent years extended into other areas. One of these is Spring MVC [?] which aims for a cleaner separation of the model, view and controller than Struts 1 with extensive linking to the original Spring framework to structure the business logic. This leads to a much cleaner MVC implementation but still requires application code to implement and extend quite a lot of framework classes [?].

3.5.5 Java Server Faces

While all previously presented frameworks were request/response-based, Java Server Faces (JSF) [?] takes a higher level approach using event-based components which more resembles desktop-based applications. JSF was developed by Sun Microsystems in 2004 as an offical standard defined in JSRs 127 and 252, with the aim of focussing development effort much more on the auctal business logic. Instead of directly responding to HTTP requests, user actions are mapped to events which are given to application code then altering a component tree representing the UI; most of the control flow is defined with long XML stanzas. This causes a very rigid programming model where one-off deviations are nearly impossible and extensions to the pre-defined components have to be very clearly separated. For small, simple web-based applications with the user experience designed around the classic hypertext paradigm instead of desktop-like behavior, the current JSF series is considered overkill. The recently finalized specification of JSR 314 defines the JSF 2.0 series and reacts to the criticism of being overly complicated, providing easier configuration and more pre-defined components.

3.5.6 Wicket

Wicket is a component-based framework initially developed in 2004 on Sourceforge.net, then moved to the ASF [?]. Central motivations where the weak support for managing server state in existing frameworks, reluctance to the JSP approach of embedding logic in the view, and the notion that current frameworks at that time were too complex [?].

With its strong focus on doing as much as possible in code, its testability suffers from very tight integration with the framework, as almost all parts of the application have to extend framework classes. Being component-based, it requires the construction of a component tree which also results in a lot of code all for the sake of keeping the view free from logic. To further that goal, it uses non-standard templates mixing HTML and a kind of custom tag library and non-standard HTML attributes without any reliance on or interoperability with JSTL, preventing the reuse of any existing tag library.

3.5.7 WebWork 2.x and Struts 2

WebWork was developed by the OpenSymphony project with "developer productivity and code simplicity in mind" [?]. To achieve this goal, it consistently refined many of the most cumbersome and verbose parts of Struts 1 and introducted refreshingly simple solutions for Struts' shortcomings [?]. In 2005, the development teams of WebWork and Struts of Apache Jakarta joined their effort to produce the Struts 2.x branch [?]. The remainder of this section will cover both frameworks as they share the majority of concepts and functionality.

The most fundamental changes in WebWorks were the reduction of dependencies in application code on the framework and a streamlined programming model shifting the workload away from the developer onto the framework.

In Struts 1, action classes had to extend a framework class, preventing any inheritance to be used by the application code for its own purposes. WebWorks replaced this by a mandatory Action interface and further optional interfaces to get additional specific helper objects injected, e.g. for a map of session properties or the underlying request object. There are no mandatory references to the Servlet API increasing the testability of WebWork actions. Struts 2 supports even a completely Plain Old Java Object (POJO) as action, without any dependency on framework interfaces or classes.

To simplify the programming model, WebWork creates a new instance of an action class on every request. This both relieves the developer from having to write thread-safe code and at the same prevents him from putting too much business logic in the web-layer – there should be a separate business layer to do the actual work. Also, request parameters are directly bound to action properties by means of deep property binding. This approach doesn't require the code duplication criticized of Struts 1's `ActionForm`s by supporting the use of actual entity classes from the domain model in a WebWork action.

Along with support for deep property binding, WebWork provides extended type conversion from the purely string-based domain of HTTP to Java objects and advanced validation routines. In every step from request

processing, control flow, type conversion to validation, the framework can easily be extended through interfaces, often without explicit configuration.

A disputed point was the custom-developed Object Graph Notation Language (OGNL) which is used mainly in WebWork's view components like JSPs. While it enables deep property binding and thereby is one of the central strengths of WebWork, it isn't officially standardized like the JSTL EL and is still lacking full generics support which makes developing against a Java 1.5 domain model unnecessarily complicated in specific cases [**?**]. It's overly broad application in the framework was also the cause for a security vulnerability [**?**].

3.6 Charting Frameworks

To easily provide visually pleasing charts for key elements of the SPQR data model, the JFreeChart framework [**?**] is used. It supports a multitude of different and highly customizable chart types, factory methods and classes for quickly producing results and through its extensibility allows to wrap existing parts of the data model as data sources without modification.

4 Implementation

4.1 Choice of Framework

The choice of framework should consider all of the points given in section 3.4 surrounding the ease of learning and development as well as maturity and future proofness. Considerung the possible framework candidates, the requirements on the web-based front end and the nature of the existing data model and solver logic, the choice fell on the Struts 2 framework.

Given the amount of documentation available as guides, manuals, wikis, tutorials, and articles on both the project's own website as well as many third-party knowledge portals and the availability of books written by the main contributors of Struts 2, both easy entry to the framework and advanced in-depth studies are provided for. Being one of the first and most widely used MVC frameworks, its concepts are known to every developer of web-based applications and knowledge from previous work in the web-application domain can be reused through the reliance of Struts 2 on many standard libraries.

In actual development, Struts 2 concentrates on solving the most recurring and complex problems for the web-application developer while staying out of his way as much as possible. Most common application code does not

have to implement any interface or extend any class of the framework itself and doesn't even rely on the Servlet API. This allows for a very high degree of testability as any required properties of an action can be injected. Sane defaults and intuitive conventions keep configuration to a minimum while still providing means to quickly change settings. With its focus on being a web-framework Struts 2 explicitly stays clear of the busines logic leaving it to other frameworks proven in that domain like Spring or EJB3.

Owning to its nearly ten year heritage, its continued development and production use provide the Struts 2 team with a lot of insight on how a web-based application framework should support the developer. Since the beginning, the project had defined bug tracking and release management with a lively community discussing and influencing the further development of the framework. New releases are made available early before the general availability release and contain upgrade instructions listing known incompatibilities and deprecation notices. As of November 2009, there is ongoing active development and future releases are planned reaching into 2011 considering current release cycles [**?**].

For the development effort at hand, Struts 2 provides some unique features to speed up the implementation of a web-based front end:

Deep Property Binding With the data model of SPQR being deeply nested and the idea of displaying an intuitive overview of the data, deep property binding as supported by Struts 2 takes much of the work of accessing and setting entity properties from the developer. With the addition of a few Java Bean conformant constructors and property accessors/-mutators, no changes had to be made to the existing data model to directly use it with the framework. This allows both a very expressive and elegant display of data in the view as well as nearly no effort on part of the developer when setting data from the view back into the data model.

Minimal Dependencies The amount of code required to run a web application with Struts 2 is minimal allowing for very rapid development. The near lack of any mandatory dependencies allows for easy refactoring to achieve the highest level of reuse of application code without framework dependencies inhibiting the architecture. Once used to the dependency injection and convention over configuration used by Struts 2, the elegance and expressiveness of the application code compared to any other of the candidate frameworks surveyed is striking.

Easy File Upload To input existing scenarios or previously solved results to the web-based front end, a file upload has to be performend via

HTTP. While being easy to implement on the client-side in HTML, the server-side usually requires lots of code and has to handle many error conditions, e.g. upload size limits, temporary storage of uploaded data, or handling users submitting an empty form. With file upload being a very common and cumbersome task in web-based applications, Struts 2 put a lot of effort into it resulting in file upload being as easy as having a property setter with a `java.io.File` parameter.

4.2 Code Structure

As one of the requirements was being a simple and lean web-based front end around the existing data model and solver system, the actual code structure of the web application is very shallow. Functionality of the Struts 2 framework has been used wherever possible with the application code written only mediating to the existing SPQR system.

The overall structure consists of various action classes implementing application logic, two custom type converters, several JSPs representing the view and three configuration files for the web application, Struts, and Tiles, a templating system.

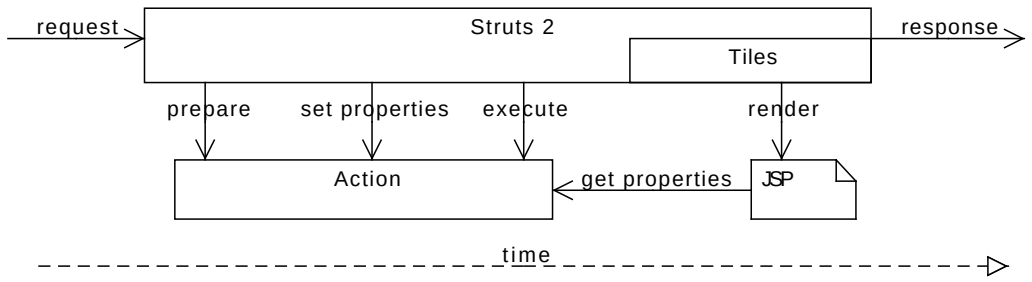


Figure 6: Control Flow in a Struts 2 web application between the framework, actions written by the developer, the Tiles templating system and JSP view components.

In Figure 6 the general flow of application logic between the Struts 2 framework and actions written by the developer is laid out on a timeline: Upon receiving a request, the framework creates an instance of the action class mapped to the URL and invokes the prepare method if available. If there are request parameters, they are set onto the action using standard Java Beans conformant property setter methods using automatic type conversion and deep property binding. If configured, validation of properties

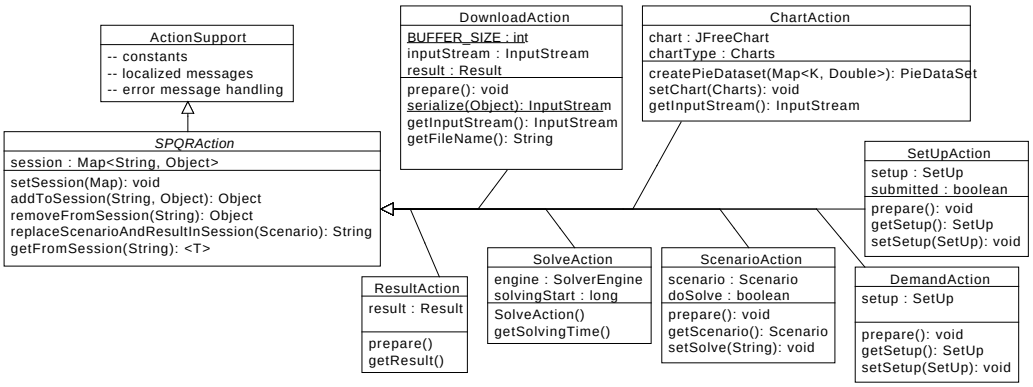


Figure 7: The structure of action classes that implement the web-based front end to SPQR. Except for the abstract class `SPQRAction`, every action implements a method with signature `execute(): String` invoked by Struts 2 not shown here for clarity.

is handled in this step and error messages are generated if necessary. After all properties have been set and optional validation has passed, the execute method is invoked granting control to application code. Upon termination, this method returns a string naming a result defined in the Struts configuration file `struts.xml`. This result can either be another action, a plain JSP or HTML document, the name of a template (e.g. provided by Tiles) or a special result class like `StreamResult` which provides means to download binary data as the output of an action.

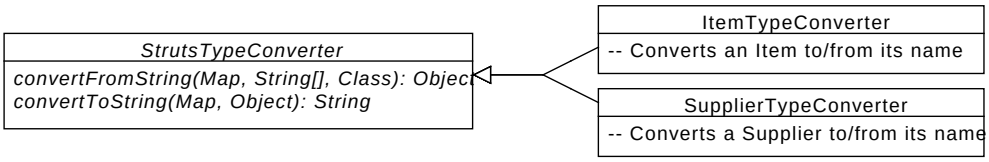


Figure 8: Two custom type converters for the Struts 2 framework reduce the amount of code for the use case of creating and editing a set up. With these two converters, the `SetUpAction` and `DemandAction` can directly work with `Item` and `Supplier` objects instead of having to manually resolve their names to objects in the data model.

The action classes all extend from a single abstract class `SPQRAction` providing easy and type-safe access to the HTTP session where the data model loaded or created by the user is stored for the duration of his work with the web-based front end. Although not mandatory, the `SPQRAction` extends

Struts' `ActionSupport` class to allow subclasses access to various constants, the message bundle used for internationalized text and the error message mechanism of Struts 2. This could be achieved without `ActionSupport`, but would require a considerable amount of code.

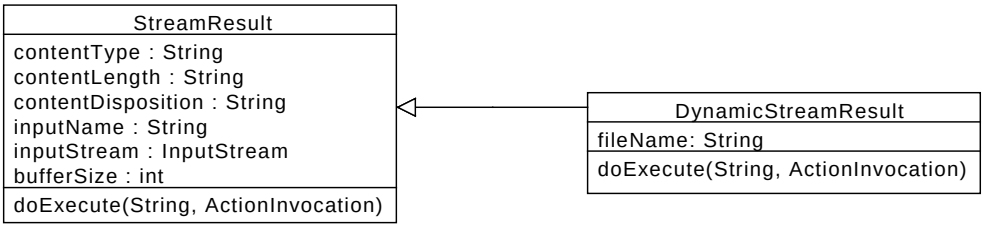


Figure 9: The custom implementation of a result class with support for dynamically setting the file name of a download.

Although Struts 2 already provides a `StreamResult` class to allow actions to create a response consisting of binary data by exposing a `InputStream` property, the configuration of this feature is limited to defining the property name, buffer size and HTTP content type of the response. To allow the end user to download a modified SPQR scenario or a solved result for later inspection, the ability to dynamically specify the HTTP content disposition header was missing. This allows to provide a file name to the browser consisting of the name of the scenario at hand and a custom file extension to easily identify which files belong to the SPQR application and what their contents might be. To achieve this, a custom `DynamicStreamResult` extending `StreamResult` was implemented, adding the requirement that each action class using this result provides a method with signature `getFileName(): String`. This custom result class is used with the `DownloadAction` in figure 10.

4.3 User Interface Design

For a consistent look, all pages given out to the user are rendered using the Tiles framework, a simple composition-based templating system. The content of each page is wrapped in a general layout which is then pleasantly styled through the use of a global CSS stylesheet. This allows the actual page contents to focus on its task without getting lost in layout details.

4.4 Visualization of Key Elements

With the end user invoking the SPQR system being the weakest link in complex bidding processes, the amount of support towards helping the user

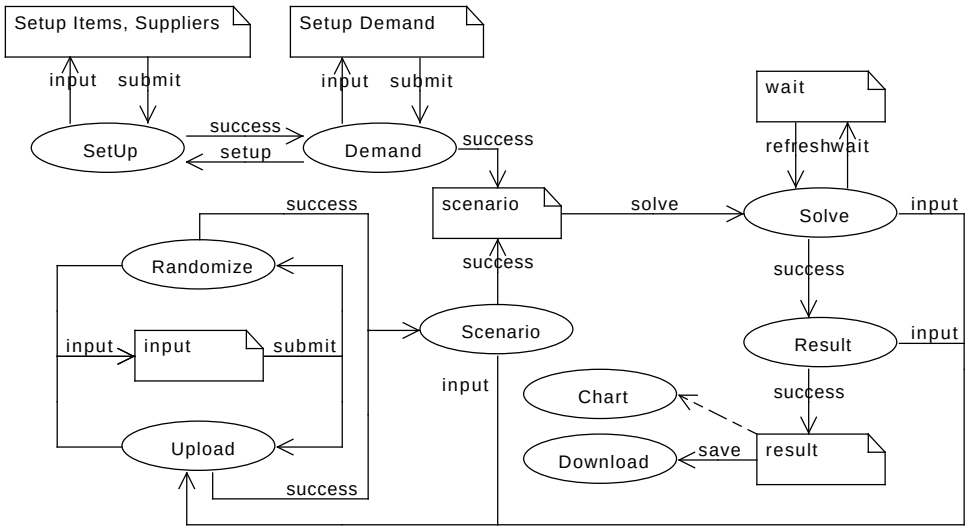


Figure 10: Complete control flow between all pages and actions. The dotted arrow to the Chart action signifies inclusion in the output of the result page. For clarity, direct navigation possibilities using the menu bar are not shown.

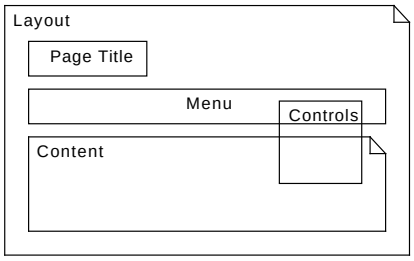
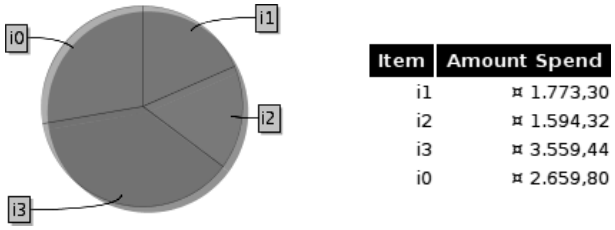


Figure 11: The general page layout as implemented with Tiles consists of a JSP providing the overall HTML structure and menu system as well as including the respective page title and content JSP for every page. Additional per-page controls are displayed on the right-hand side.

quickly and fully understand the large amounts of data is crucial. To provide this level of support, key elements of the data model are visualized both in tabular and graphical form to allow the user to reduce the cognitive complexity of the auction process recognized in [**?**], page 14.

Being a especially complex and important part of the auction, the offers submitted by suppliers along with their payment modifiers and discount



Item	Amount Spend
i1	¤ 1.773,30
i2	¤ 1.594,32
i3	¤ 3.559,44
i0	¤ 2.659,80

Figure 12: An example chart displaying the spend per item along with the original tabular data. Labels a directly attached to the sections of the pie chart for easy referencing with the data table.

conditions are the most demanding but at the same time worthwile elements to visualize. With SPQR supporting both incremental and overall payment modifiers affecting the price of an item depending on the amount purchased, each offer is rendered in a chart containing all payment modifiers of an offer.

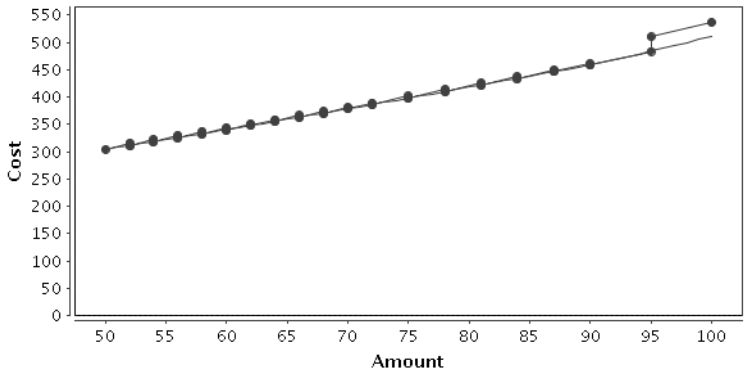


Figure 13: An example of the visualization of an offer with *incremental* bidding.

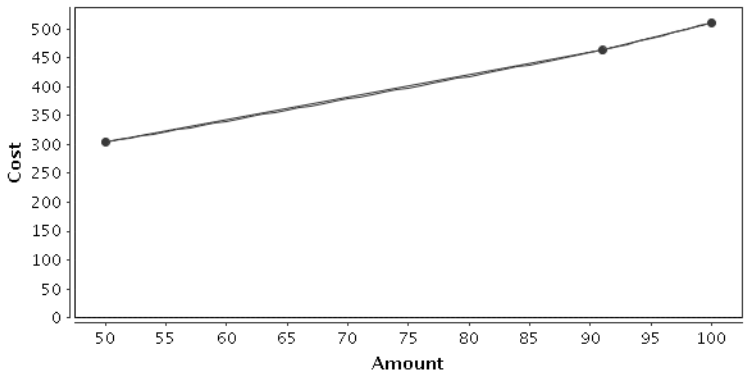


Figure 14: An example of the visualization of an offer with *overall* bidding.

With incremental bidding, a supplier bids on a bundle of items by submitting an offer with payment modifiers for short amount ranges, resulting in the jigsaw pattern of figure 13 based on the actual cost function for the supplier. This is in contrast to overall bidding, where the supplier matches his cost function with a usually smaller number of linearly interpolating payment modifiers, as seen in figure 14. Both visualizations chart amount of money spend on the horizontal and cost on the vertical axis, displaying the submitted bids on top of the supplier's actual cost function.

4.5 Issues

Several issues were encountered during the implementation. Among the most pressing ones was the availability of the various native solvers used for optimizing the scenario. As they are written in C/C++ instead of the Java Programming Language, each required multiple libraries at runtime that were not readily available on all target platforms. Often the solver would crash or deliver unusable results due to subtle changes in the SPQR data model or fields with `null` value. In the course of this thesis, three different solvers were tested in the backend, finally settling on the Gurobi Solver [**?**] available pre-compiled for both Windows and Linux platforms.

To more easily switch solvers and even allow runtime changes, a minimal solver interface based on Remote Method Invocation (RMI) was developed as shown in figure 15. It encapsulates the different ways to create and call the solvers from the the Java Programming Language with a simple interface and transparently provides both local and remote access to a solver implementation.

The semantics of OGNL and its full application as well as various pieces of information about the inner workings of Struts 2 were only made clear in the book [**?**].

Various minor bugs in the framework, especially with OGNL and generics in the data model, could be fixed by upgrading to the latest available release of Struts 2 during the development of the code.

5 Conclusion

This thesis provides a thorough analysis and solution for the usability problems of the command-line based SPQR application building on best practices of current software development. By providing a web-based front end, the handling and communication of important aspects of the bidding process is dramatically improved. Through the comprehensive presentation of the

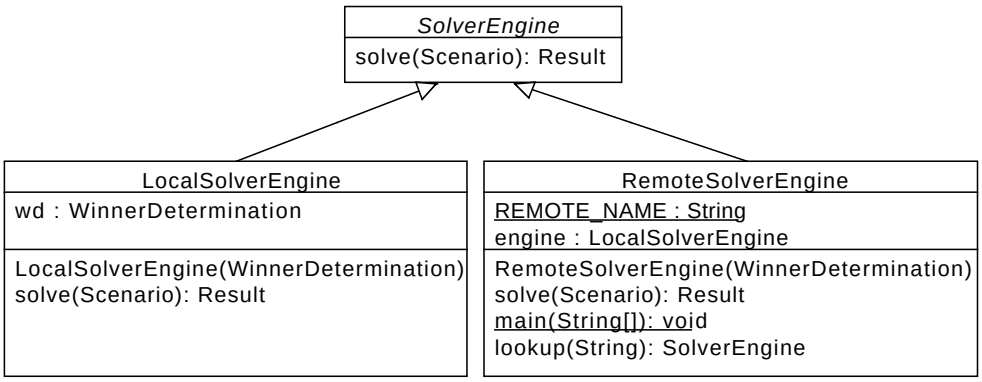


Figure 15: The solver engine approach to decouple the caller from the executor of a solving request consists of an interface `SolverEngine` and two implementations for local and remote invocation, the latter using RMI. To be as compact as possible, the `RemoteSolverEngine` implements both the required interface and provides a `main` method to run it as an application.

various means for implementing a web-based application in the Java Programming Language and a deliberate and quantifiable approach to choosing a framework through the application of metrics for the learning, development and maintenance phase of a project, a front end to the SPQR system has been developed that implements all requirements with a minimum of elegant and clean code while allowing easy extension and upgrades in the future. To effectively assist the end user in grasping the complex bidding process, key elements are visualized in an intutive way cutting down the large amount of numerical data the user has to digest. While focussing on allowing the end user to much more easily use the functionality that SPQR provides, the choice of framework, programming style and extensive investment into following programming best practices wherever possible enable future enhancements to the web-based front end to be implemented quickly and cleanly.

A Setup and User Documentation

A.1 Setup Instructions

This section describe the setup and operation of the SPQR and its web-based front end. It should run on any platform supported by the Java Programming Language and the desired solver component.

Requirements

The following requirements are mandatory to operate SPQR itself:

- Java Runtime Environment (JRE) version 1.5 or later
- Windows, Linux, or Mac OS in a version recent as of 2009
- The binary artifacts (.dll, .so, .lib) of the desired solver for the respective operating system including potential additional dependencies required by the specific solver.

The following *additional* requirements are mandatory to operate the web-based front end to SPQR:

- A web application container or full-fledged Java Enterprise Edition (JEE) application server supporting at least the Servlet 2.4/JSP 2.0 specification

The web-based front end to SPQR was sucessfully run in the following environment:

- Ubuntu 8.04.3 LTS
- OpenJDK 1.6.0_0-b11
- Gurobi Solver 2.0.1
- Tomcat 5.5.25-5ubuntu1.2

Instructions

- Follow the preferred way of software installation for your operating system to install the JRE, web application container or application server, the desired native solver, and all its dependencies

- Compile the Java web application containing the web-based front end to SPQR into a single Web Application Archive (WAR) file and deploy it to your web application container or application server. Note to adjust the

- Optionally: To allow runtime changes to the solver or run the solver on a different host than the web-based front end, change `SolverAction` to acqurie its solver engine through `RemoteSolverEngine.lookup(host)` with `host` being the name or address of the system running the actual solver engine.

  On the host to run the solver engine, run the `RemoteSolverEngine` class with a classpath containing the libraries from the web application and pointing the VM property `java.library.path` to the binary artifacts of the desired solver.

A.2 User Manual

The the web-based front end to SPQR is accessible with any recent browser. In case of additionally configured access restrictions, contact your administrator.

On the top of every page you will see the title followed by the menu bar as shown in figure 16. Those two elements are visible on all pages and allow easy navigation between the various screens. Screens not currently available or applicable to the application state are not shown. Some pages will present you with additional controls or quick references to specific parts of a longer page at the right hand side. When scrolling down a long page, these controls will stay visible at all times for easy access.

Inputting data to SPQR through the web-based front end happens on the "Upload a Scenario" screen shown in figure 17 where you can either upload an existing text-based scenario description, previously loaded and saved scenarios and/or results and also create a randomized scenario for testing purposes. After providing a scenario, you will be given the chance to review its contents.

When you want to solve a scenario, click the "Solve" button in the controls of the "Review and Refine Scenario" screen after you have loaded or created a scenario. A wait page will be displayed to you while the scenario is being

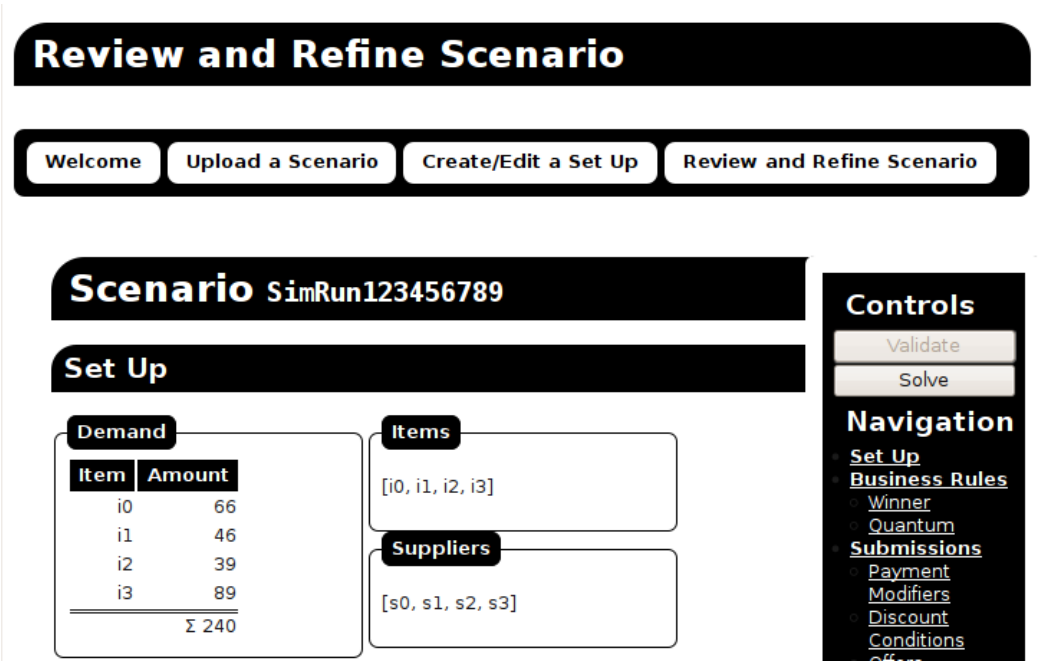


Figure 16: The UI of the web-based front end to SPQR.

solved.

After the solver has finished, you will be shown the result screen where you can inspect the result and optionally download a file containing the exact scenario and result for later reference so you don't have to wait for the possibly long solving process again.

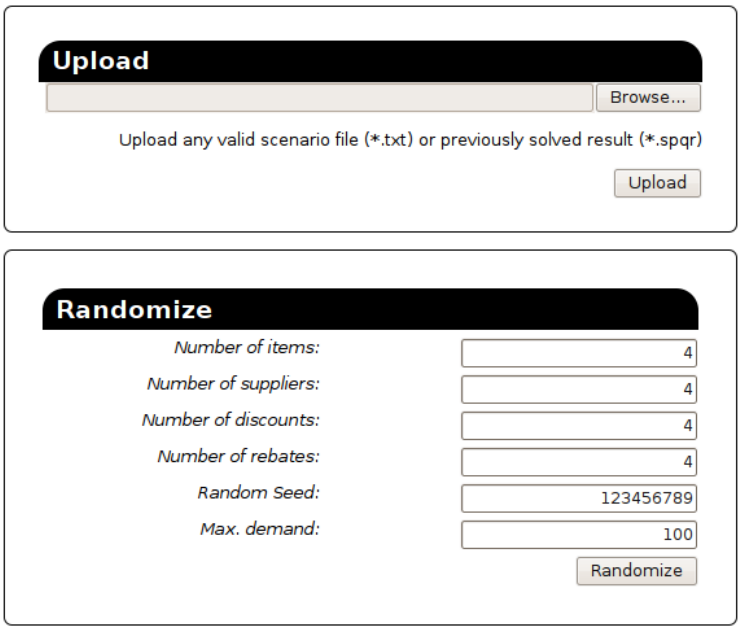


Figure 17: The input mask of the web-based front end

List of Listings

List of Figures