A General Web-Interface of an Assessment-Engine Instantiated for a TP-based Mathematics Assistant

Master Thesis
at IICM
Graz University of Technology

Thomas Zillinger
tzilling@sbox.tugraz.at
April xx, 2014

Supervisor:
Univ.-Doz. DI Dr.techn. Christian Gütl,
Graz University of Technology

Co-Supervisor:
Dr.techn. Walther Neuper,
Graz University of Technology
Abstract

This master thesis combines research and development on specific novelties in web based technologies with a specific application of such technologies. The application concerns two academic research prototypes at two different institutes at TU Graz, IICM and IST.

Research on front-of-the-wave web based technologies preceded the practical work within the master thesis. The results of this research have been documented as a part of a seminar/project. The results were a feasibility study with a positive conclusion. The conclusion comprised an architectural design and a detailed design of how to accomplish the challenges posed by the envisaged application of the web technologies. The application is a general web-interface of an assessment-engine for a TP-based mathematics assistant.

So, this thesis describes a connection of two different systems. (1) a "Modular Assessment System for Modern Learning Settings (MASS)" developed at IICM. MASS is generic in that it abstracts assessment items such that various e-learning tools can use its services; thus also the web interface shall be generic and support physical connection of MASS to a variety of e-learning tools.

(2) ISAC ("ISAbelle for Calculations in applied mathematics") developed at IST, a mathematics assistant, which supports step-wise solving of engineering problems similar to paper-and-pencil calculations, but with automatically generated feedback. Isac is based on the Theorem Prover "Isabelle" and thus resides on servers with respective resources.

The development and implementation of the web-interface is described in detail. In order to cope with the complex functionality of both, MASS and ISAC, development starts with user stories, which are broken down such that they are amenable for implementation and also serve handling of the systems. The proceeding towards concrete implementation validates the design from the feasibility study; all intricacies of the various technologies involved can be overcome, sometimes in very unique and creative ways.

Ample space is given to descriptions of how to setup and handle the various technologies and systems: Isabelle, Isac, the version control system (Mercurial), the integrated development environment (NetBeans), etc. These descriptions are provided as a reference manual for both, for developers continuing on MASS and on Isac, as well as for researchers, who are going to apply the web-interface in field tests.
**Statutory Declaration**

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

April 12, 2012

date signature
Contents

1. Introduction
2. Web Interface Technology
   2.1 Clients and Servers
      2.1.1 Communication between applications
      2.1.2 Web Services
   2.2 Web application frameworks
      2.2.1 Java Server Faces
      2.2.2 Struts
      2.2.3 Java Web Start
   2.3 Evaluation of Components for a Web Service
      2.3.1 The Use Cases for Applying Web Technology
      2.3.2 Evaluation with respect to the Use Cases
3. Architectural Design
   3.1 Architecture of MASS’s Web Service
   3.2 ISAC’s Architecture as an Example
   3.3 User - Application Lifecycle
   3.4 Decision for the particular web technologies
4. Detailed Design and Implementation for ISAC’s Web Interface
   4.1 The Implementation process starts with User Stories
   4.2 Setting up ISAC
   4.3 User Stories for the new functionality
   4.4 Converting and working with Mercurial
   4.5 Creating a virtual appliance for Isabelle, the Theorem Prover
   4.6 Making ISAC work as a Netbeans project
   4.7 Configuring ISAC and Netbeans
      4.7.1. Install MySql via xampp for the UserLogger table
      4.7.2. Configure ISAC to connect to Isabelle via ssh client
      4.7.3. Configure ISAC
      4.7.4. Configure the application server
   4.8 Starting ISAC for the first time
   4.10 Identification and Start of an Assessment
      4.10.1 Status-quo in the ISAC Maths Assistant
      4.10.2 Detailed Design for the Instantiation
   4.11 Execution of Assessment Items
      4.11.1 Status-quo in the Maths Assistant
      4.11.2 Detailed Design for the Instantiation
   4.12 Enforce the Assessment mode
   4.13 Summative Assessment
      4.13.1 Status-quo in the Maths Assistant
      4.13.2 Detailed Design for the Instantiation
5. Summary and Conclusions
Glossary
References
1. Introduction

The World Wide Web provides a broad range of technologies, influential both in the private and public sectors. This master thesis addresses an educational application of the WWW and combines two developments at the Graz University of Technology.

At the Institute of Softwaretechnology (IST) an experimental mathematics assistant, called ISAC, is under development\(^1\). ISAC’s experimental character is given by its novel basis, (Computer) Theorem Proving (TP), while other educational mathematics assistants prefer Computer Algebra Systems, which have already been available for a long time. TP technology is becoming more and more relevant in software engineering at this time: this technology allows one to check critical features of software, for instance whether or not an internet connection for home-banking is safe against certain kinds of attacks. ISAC takes advantage of the innovation window for TP technology and uses the TP Isabelle [NPW02] for experiments towards a novel generation of TP-based educational mathematics assistants [Neu12, Neu13].

At the Institute for Information Systems and Computer Media (IICM) a Modular Assessment System for Modern Learning Settings (MASS) has been researched and developed. According to [Gütl07, Gütl08] is a MASS characterized by: (a) a flexible design to be used as a stand-alone system or to be easily extended by third-party tools. (b) user-friendly interfaces for both students and educators in which a user interaction and online submission of solutions and evaluations can be done. (c) an assessment environment for various learning and assessment settings which supports guided as well as self-directed learning. (d) a management and (semi-)automatic support throughout the entire assessment life cycle (exercises creation, storage and compilation for assessments, as well as assessment performance, grading and feedback provision). (e) a rubrics design and implementation interfaces to allow the educators to design their own rubrics based on learning objectives to assess learners’ performance against a set of criteria. (f) a support of various educational objectives and
subjects by using various tools sets which, for example, enables automatic exercise generation or selection, automatic grading and feedback provision. (g) a results analysis and feedback provision (immediately or timely) of the current state of user knowledge and meta-cognitive skills for both educators and learners and also for adapting course activities and learning contents based on users’ models. (h) standard conform information and services to be easily sharable, reusable and interoperable. This may include the tests’ questions, answers and students’ results, rather than any other required services. And finally, (i) security and privacy where a secure log-on of users based on pre-defined levels of access, and also users’ authentication based on machine (domain users) or by usernames/passwords.

These two developments of IST and of IICM shall be connected by a web based technology. The choice for using web based technologies is motivated by the demand for ubiquitous access to learning resources. A premise for the planned connection via the Web is given by the decision to start from the side of assessment: A MASS is a general tool for various kinds of learning scenarios as mentioned above; it usually is to be connected to a multitude of learning tools, ISAC is only one example of such a learning tool. The goal in this master thesis is to create a general web interface for a MASS; in particular the connection to the learning tools is to be designed such that it can be instantiated to a particular tool with minimal effort. The connection to ISAC shall be an exemplary instantiation of the general connection.

The task of this master thesis is to (1) identify the concepts, technologies and components required for the web interface for a MASS and (2) design and instantiate the web based connection to ISAC.

Point (1) reuses the results of a feasibility study [Zill2013], which involved a review of the respective literature on the concepts and technologies, as well as retrieval of vendors for technologies and components. It also involves the evaluation of respective features, considering several combinations, which finally lead to a well founded selection of components, in turn allowing for an architectural design for the web based connection between a MASS and a
certain class of learning tools, ISAC as an example.

Point (2) concerns the work within this thesis. The design envisaged in the feasibility study is refined by "user stories". These reflect the users requirements and support the development process and give a clear vision of the overall goal. Also detailed descriptions of the build, setup, implementation and assembly process of the product are provided.

This point is different to (1), since the architecture of ISAC is already established and documented, and ISAC’s technologies and components are already selected and implemented. Therefore this work starts with an analysis of ISAC’s architecture with respect to several variants of web access. It proceeds with an evaluation of the variants which lead to the decision for a specific design. This design is aware of ISAC’s state of development as an experimental prototype, which implies simplicity and adaptivity. An implementation of the design provides a proof of concept, probably efficient enough for small field tests.

This written protocol is structured according to the rules for a master thesis: One half has to address scientific research and another half has to address a practical project. § 2 addresses scientific research on specific web based technologies, § 3 and § 4 address the practical project.

§ 2 researches the state-of-the-art in web interface technology, focuses on variants of client-server technologies in § 2.1 and web application frameworks (JSF, Struts, JNLP) in § 2.2. In § 2.3 the research results are evaluated and reviewed in relation to three use cases, which combine a MASS and ISAC in a novel way.

§ 3 presents the decisions for an architectural design on web interfaces for both, for a MASS in § 3.1 and for ISAC in § 3.2. The implementation of ISAC’s web interface is the basis for further development and thus described in appropriate detail. § 4.10 describes how a connection between a MASS and ISAC is established, § 4.11 explains handling of assessment items and § 4.13 addresses the assessment at the end of a session. Finally, § 5 presents a summary and conclusion.
2. Web Interface Technology

The focus of our research interests are interfaces between existing software systems that are connected over the World Wide Web\(^2\). These existing systems pose significant challenges for web interface technologies, in particular for ISAC: The ISAC project aims at convenient and simple interaction of learners with a mathematics assistant during stepwise problem solving - the kind of problem solving, which has traditionally been done with a pencil and paper, for instance during a written exam. This master thesis comes up with a surprising solution, which might be completely different in two years, due to the rapid development of web technologies.

Thus we begin with a survey on specific technologies which might be appropriate to accomplish the goals of interactivity between ISAC and a MASS.

2.1 Clients and Servers

![Figure 2.1a: Typical Client Server Infrastructure](image)

“The client/server model is a computing model that acts as a distributed application which partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.”\(^3\)

The client/server model is fundamental in computer science [Elb94].
With this client/server model, the client and the server can be situated on the same machine or on different machines, either communicating within the same program/memory space or over some network, like the Internet.

The Internet mentioned here had its origins in the 1960s\(^4\). The ARPANET and other projects led to the development of essential protocols that enabled networks to be created which consisted of other networks themselves. Using this knowledge, the ARPANET grew quickly from 15 networks in the 1970s to what is called the ‘Internet’ today. On this road to growth, several other protocols were standardized, like TCP/IP\(^5\) in the 1980s which "provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination."\(^6\) In the application layer\(^7\) of this 4-layer protocol, several other protocols for various applications are used. For example, for e-mail the SMTP, IMAP or POP3 protocol are used, which were first defined in the 1980s. Another fundamental application layer protocol is HTTP\(^8\) - the Hypertext Transfer Protocol. Within the client-server computing model\(^9\) the HTTP protocol functions as the request-response protocol. On one computer an application like a web browser\(^10\) acts as the \textit{client} and somewhere else on the Internet on some other computer another application could host a website that acts as the \textit{server}. The client submits a HTTP request, which is received by the server that interprets the request and sends back a response to the client. This response can be any electronic resource, like HTML\(^11\) pages which are viewed in a web browser or files which are used in other applications on the client side.

There are basically two options an HTTP request can serve:

- A client may request data from the server
- A client may upload data to the server

However, a connection may only be initiated by the client. A server in an HTTP network is not meant to initiate a connection to the client. Furthermore, the HTTP protocol is stateless, which means that each request is treated completely independently and unrelated to the previous one.

These two properties keep the protocol simple to use. In § 2.2 web
In the scenario that is looked at in this master thesis (see § 1) there are several layers of communication:

a. A person uses a browser to access information from a web application server - the MASS.

b. The MASS website displays a link which calls an external service. In this case the ISAC frontend should be started. Additionally, some personalized information is needed to transfer to the ISAC client so that the user can begin the correct exercise when the ISAC client starts.

c. The ISAC client is basically a thin client that is remotely controlled by the ISAC server to fulfill and evaluate every step during the assessment.

d. When the user finishes the exercise, he returns to the MASS client to continue the assessment. The MASS server needs to retrieve the
results from the ISAC server in order to evaluate the results.

In this scenario at least 4 ways of communication can be found. Various protocols are used to establish the connection and transfer information:

- In case (a) at least HTTP is used between the web browser and the MASS server.
- In case (b) a HTTP GET request is made to call the ISAC client, which is in turn downloaded.
- In case (c) the Remote Method Invocation protocol (RMI) is used where the ISAC client reports every click and piece of information that is entered in the user interface (UI) to the ISAC server. The server itself controls every aspect of the client’s UI and transfers back information on how the UI should react or change.

Cases (a) - (c) are basically predefined by the servers and clients that are being currently used, since these are already existing applications.

In case (d) the MASS server wants to retrieve information from the ISAC server. There are several ways to create some kind of a web service to do exactly that.

### 2.1.2 Web Services

“Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.”

[WSA12]

A web service that is defined by W3C must first define its own interface. This interface is a contract between a server and a client. It is described in a machine readable XML file called the Web Services Description Language (WSDL).

An example for a such a WSDL could as follows:

```xml
<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsd1/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsd1/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
```
According to this example the web service is called HelloService and provides one operation sayHello. The server implementing this web service receives one string parameter firstName, which is defined within the sayHelloRequest, and returns one string value, which is defined in the sayHelloResponse. The binding defines the transport protocol as SOAP HTTP. The whole service can be reached at the following URL: http://www.examples.com/SayHello.

To implement the actual functionality on the server side or the client side,
only the WSDL contract is needed. Complete clients for such a web service and initial stubs for the server functionality in the target programming language (eg Java, .NET) can be created automatically, which presents a major advantage over other web service technologies.

However, recent developments [PW12] show that more lightweight approaches are becoming more popular. Their major advantage is that they are comparably easier to design and to implement since there is no need of an interface definition in form of a WSDL file. Additionally, simple protocols than SOAP are used. These Web APIs are used most notably in Web 2.0\textsuperscript{17} client side applications. They are based upon the representational state transfer\textsuperscript{18} (REST) protocol and are called in short RESTful services. Whereas a typical WSDL SOAP request would closely resemble the following code\textsuperscript{19}:

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:body pb="http://www.acme.com/phonebook">
       <pb:GetUserDetails>
           <pb:UserID>12345</pb:UserID>
       </pb:GetUserDetails>
   </soap:Body>
</soap:Envelope>
```

an equivalent REST request would look like this:

```
http://www.acme.com/phonebook/UserDetails/12345
```

A more complex REST request could then look something like this:

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

It is common to use GET requests (like these above) for read-only data retrieval services exclusively and POST requests usually for data manipulation. However POST requests could always be used for both, for example if a parameter contains binary data, which would be too long for a
GET request.

Contrary to WSDL SOAP requests which use only XML, several content types can be used in REST server responses. [ME12] states that XML and JSON are most commonly used for REST responses, although other types like CSV or even HTML are also possible. Basically there are no limits to the response type, as long as the client can interpret it.

XML is used wherever type safety is important, CSV is very compact and JSON is easy to parse by JavaScript clients like a common web browser and is often used in combination with AJAX technologies.

A web browser could, for instance, show a webshop with different offers. Whenever the user clicks on the ‘next’ button, the website shows another batch of offers without reloading the whole page. In this example the ‘next’ button calls a JavaScript method, e.g. `loadNextOffers()`, which in turn makes an AJAX call to a REST service URL to the web service provider. The server then returns a JSON response which could look something like the following example:

```json
{
   "currentPage": 2,
   "totalPages": "15",
   "elements": [
      {"id": "21", "name": "Flower bouquet small"},
      {"id": "22", "name": "Flower bouquet medium"},
      {"id": "23", "name": "Flower bouquet big"}
   ]
}
```

Within the web page the JSON response will automatically be converted into JavaScript objects. Finally, another JavaScript method would only need to convert the response into HTML DOM objects in order to display the items on the web page.

### 2.2 Web application frameworks

“A web application is an application that is accessed by users over a network..."
such as the Internet or an intranet.”

Modern day web applications have grown more sophisticated than the HTTP protocol is able to provide features for (see § 2.1). For instance, a Web Shop application needs to remember the items a user wants to purchase. Another web application, however, needs to authenticate a user and thus authorize him to access certain content of the application. These and other cases need to remember states between interactions of a user. Since these cases are not directly supported in HTTP via stateful session support, other methods are being used to accomplish session support. These methods use e.g. HTTP cookies or GET request query-strings and server side session, or hidden variables in HTML forms.

It can be observed that in today’s web applications the communication between client and server are becoming more and more intertwined. This can be observed by how closely web applications are growing toward Rich Internet applications (RIA), whose characteristics more closely resemble those of desktop applications rather than simple static web pages. The former have rich possibilities of user interface interaction while the latter can
also just be a collection of hyperlinked HTML pages.

HTTP requests from the client usually result in a retrieval of a new, full, HTML page from the server. By design, it is not possible to submit a request to which the server’s response would only change parts of an HTML page on the client. This limitation has been primarily overcome by requesting information from a server via JavaScript within the client’s web page. Depending on the server’s response, JavaScript is used again to change only parts of the client’s HTML. Usually AJAX technologies are employed to do so.

Other limitations like the inability of a server to contact the client to notify a user in the web browser have also been circumvented. In this case usually AJAX is used on the client side to constantly poll the server for notifications, merely giving the impression that the server contacted the client.

These basic and advanced functionalities as well as many others are usually hidden from the user and developers. To accomplish these goals web application frameworks have been developed continuously throughout the years. Their main goals are normally simple use and the provision of a rich feature set for users and developers.

There are many web application frameworks available, some are purely used on the client side, others exclusively on the server side, still others mix client and server API’s transparently and conveniently for the developer together.

Examples for popular frameworks include the following:

ASP.NET, JavaEE, JSF, Struts, WebObjects, web2py, OpenACS, Ruby on Rails, Zend Framework and Symfony [Waf12].

Most web applications follow the Model-View-Control architecture [Gam95].

“MVC decouples views and models by establishing a subscribe/notify protocol between them. A view must ensure that its appearance reflects the state of the model. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself. This approach lets you attach multiple views to a model to provide
different presentations. You can also create new views for a model without rewriting it.” [Gam95].

Fig 2.2.a and 2.2.b show the process sequence of a typical MVC model. The figures show the CVM process, Model 2, which is the basis for widely spread technologies like JSP, JSF, Struts or SpringMVC.

In the following chapter we want to compare the functionality of two different MVC frameworks, namely Java Server Faces and Struts. In § 2.2.3 we also want to explain a remote invocation technology called Java Web Service. 
Start. The former two show how the ubiquitous requirement of running software is solved on the server side by making software work within a web browser. The latter shows how a client-server model can be implemented by downloading and running a part of the application on the client machine while maintaining communication with a server.

2.2.1 Java Server Faces

JSF has been developed via the Java Specification Request (JSR) by a team of international companies with the knowledge and experience of previously existing web frameworks, such as Struts, JSP. JSF is a component based web application framework which mainly defines a Java API and a basic implementation. The API has also been implemented by various other projects (eg. RichFaces, Oracle ADF Faces, PrimeFaces). A controller (the FacesServlet) is responsible for parsing the HTTP request and redirecting it to the correct XHTML page. The controller then determines (e.g. from reading the faces-config.xml file) which XHTML page and which backing bean is to be used to finally send the rendered response to the client (eg. the browser). A JSF page is mostly an XHTML page which consists of JSF components, called Facelets. These components are bound to properties of a backing bean or managed bean. Some components determine additionally what should happen in response to certain events, like mouse clicks and keyboard input. For example, the component could define that a method of the backing bean should be called after a mouse click occurred. This method in turn loads new resources from the database and then the component decides if the whole page should be reloaded or just parts of it (via AJAX).

The whole lifecycle, starting from the initial display of the page in the browser, continuing with submitting and redisplaying or redirecting to another page is managed by the controller, which is given by the framework.
A significant difference presents itself when comparing the older JSP technology with the JSF technology. A JSP page is compiled to a servlet at some time in its lifecycle, which in turn renders exactly the content of the JSP page (HTML or XML or something else) to the client. In contrast, JSF builds an object component tree from its XHTML page. Each component in the tree retrieves values from the model and stores all states related to it in the context. Additionally, event handlers or validators (if used) may be attached to each component. In the render phase each component calls the target Renderer. Thus the result of rendering one page in each lifecycle may be HTML, but if the components support it, it can be any output. Thus JSF is able to render several target pages from the same source page, including everything from desktop computers, to mobile handsets, tablets and even different types within each of these categories, such as iOS, Android, Windows Phone, MacOs, Linux, Windows, a.s.o specific UI components.

Here is an example of a small JSF page, with two boolean and one submit button. A developer needs to write a view, the XHTML page, and a model,
the backing bean.

I accept terms and conditions:  No

Subscribe me to newsletter:  Yes

Submit

Figure 2.2.1a: An example JSF Page

This is the JSF XHTML page:

```html
<h:form>
  <p:growl id="msg"/>
  <h:panelGrid columns="2" style="margin-bottom:10px" cellpadding="5">
    <h:outputText value="I accept terms and conditions: " />
    <p:selectBooleanButton value="#{formBean.value1}" onLabel="Yes" offLabel="No" />
    <h:outputText value="Subscribe me to newsletter: " />
    <p:selectBooleanButton value="#{formBean.value2}" onLabel="Yes" offLabel="No" onIcon="ui-icon-check" offIcon="ui-icon-close" />
    <p:ajax update="msg" listener="#{formBean.addMessage}"/>
  </p:selectBooleanButton>
</h:panelGrid>
<p:commandButton value="Submit" update="display" oncomplete="dlg.show()" />
<p:dialog header="Selected Values" modal="true" showEffect="fade" hideEffect="fade" widgetVar="dlg">
  <h:panelGrid columns="1" id="display">
    <h:outputText value="Value 1: #{formBean.value1}" />
    <h:outputText value="Value 2: #{formBean.value2}" />
  </h:panelGrid>
</p:dialog>
</h:form>
```

Here is the model (backing bean) for the JSF page:

```java
package org.primefaces.examples.view;
import java.io.Serializable;

class FormBean implements Serializable {
  private boolean value1;
  private boolean value2;
  public boolean isValue1() {
    return value1;
  }
  public void setValue1(boolean value1) {
    this.value1 = value1;
  }
  public boolean isValue2() {
```
2.2.2 Struts

Struts is an action based framework; the HTTP request is parsed, then processed in the Actions and in the end the controller Servlet forwards the request to the JSP which is responsible for generating the response. Struts closely resembles the classic JSP/Servlet model, in which a Struts controller reads from a file (the struts-config.xml) to determine which URL is bound to which page controller for this particular page (called Action) and which view the JSP page together with which model (called Form) is to be loaded to render the response to the client. The page lifecycle is controlled via the Action. JSP Tag libraries are used to simplify the process of creating HTML pages. The response that is rendered to the client is defined by what the JSP page contains. Thus the response type can be HTML or XML or other content. As opposed to JSF (§ 2.2.1) Struts requires a different JSP page for each response type.
As an example, consider this final rendered page:

```
User Name: Eswar
Password: ****
Login
```

The JSP Page looks like this:
```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html>
<head>
  <title>Login Page</title>
</head>
<body>
  <div style="color:red">
    <html:errors />
  </div>
  <html:form action="/Login" >
    <html:errors />
    <html:form action="/Login" >
      User Name :<html:text name="LoginForm" property="userNmae" />
      Password :<html:password name="LoginForm" property="password" />
      <html:submit value="Login" />
    </html:form>
  </body>
</html>
```

The controller looks like this:
```
public class LoginAction extends org.apache.struts.action.Action {

    private final static String SUCCESS = "success";
    private final static String FAILURE = "failure";

    public ActionForward execute(ActionMapping mapping, ActionForm form, 
                                  HttpServletRequest request, HttpServletResponse response) 
        throws Exception {
        LoginForm loginForm = (LoginForm) form;
        if (loginForm.getUesrName().equals(loginForm.getPasswrod())) {
            return mapping.findForward(SUCCESS);
        } else {
            return mapping.findForward(FAILURE);
        }
    }
}
```

The model looks like this:
public class LoginForm {

    public String username;
    public String password;

    public void setUserName(String username) {
        this.username = username;
    }
    public String getUsername() {
        return this.username;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getPassword() {
        return this.password;
    }

    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if (username == null || username.length() < 1) {
            errors.add("username", new ActionMessage("error.username.required"));
        }
        if (password == null || password.length() < 1) {
            errors.add("password", new ActionMessage("error.password.required"));
        }
        return errors;
    }
}

These are only the main files that are needed, other configurations have to be made in addition.

2.2.3 Java Web Start

"The Java Web Start (JWS) software allows you to download and run Java applications from the web. The Java Web Start software:

- Provides an easy, one-click activation of applications
- Guarantees that you are always running the latest version of the application
- Eliminates complicated installation or upgrade procedures"

[WS12]

JWS is not a web application per se, however it combines some advantages of a web application and those of a desktop application in some respect,

- it can be started via Web Link on a web page or from a shortcut on
your client computer
● immediate UI responsiveness, since it is actually a desktop application
● works also offline from the intra- and internet
● it is always up-to-date: If connected to the internet, the new version of the application will be downloaded automatically, as mentioned above
● There is no need for a relatively complex web application server, which hosts the web application. However the Java Virtual Machine (JVM) must be installed on the client
● special security measures can be taken to trust certain JWS programs and restrict access to the local computer

If you have a Java Application that is already running and can be started per command line on a local computer, it is relatively simple to create a JWS application from it [JWG12]. A JWS application consists of a .jnlp descriptor, the actual application as .jar file, and optionally other .jar files the application needs. All these files need to be hosted on a web server. On the client a Java Runtime Environment (JRE) needs to be installed. A .jnlp file could look something like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for SwingSet2 Demo Application -->
<jnlp spec="6.0+" codebase="http://my_company.com/jaws/apps"
    href="swingset2.jnlp">
  <information>
    <title>SwingSet2 Demo Application</title>
    <vendor>Sun Microsystems, Inc.</vendor>
    <homepage href="docs/help.html"/>
    <description>SwingSet2 Demo Application</description>
    <description kind="short">A demo of the capabilities of the Swing Graphical User Interface.</description>
    <icon href="images/swingset2.jpg"/>
    <icon kind="splash" href="images/splash.gif"/>
    <offline-allowed/>
    <association mime-type="application-x/swingset2-file"
        extensions="swingset2"/>
    <shortcut online="false" install="false">
      <desktop/>
      <menu submenu="My Corporation Apps"/>
    </shortcut>
  </information>
  <information os="linux">
    <title>SwingSet2 Demo on Linux</title>
    <homepage href="docs/linuxhelp.html"/>
  </information>
</jnlp>
```
When a user on the client computer clicks on a link to a JWS application in a web browser, the browser actually downloads a .jnlp file. From the MIME type of this file (application/x-java-jnlp-file) the browser detects that there is an application associated with this file and starts the javaws application of the JRE on the client machine taking the downloaded .jnlp as an argument. javaws reads the file and then basically downloads all the resources (.jar files) needed, checks for security issues, stores the application in a cache on the client machine, optionally creates a shortcut in the operating system and starts the main application that is registered within the .jnlp file.

There are very few different frameworks that try to accomplish similar goals like JWS, for example ClickOnce\textsuperscript{37} for Microsoft .NET or ZeroInstall\textsuperscript{38} for multiple Operating Systems.

### 2.3 Evaluation of Components for a Web Service

As mentioned initially, the above description of state-of-the-art web application and interface technology has been selected with respect to the intended usage of the existing software, MASS and ISAC.

In this section we make the selection of technologies explicit and review them according to use cases, which are intended to be accomplished by respective technologies. Since the above description of web interface technologies already includes the point of view of web users, the review with respect to the use cases is straightforward.

To find out which approach is to be taken to connect an existing MASS with the ISAC engine, the main scenario needs to be described and from this it is
A group of students are enrolled in a course like ‘Introduction into Mathematics’ at a university. The final exam will be taken electronically. The professor who organises this course creates an exam by putting together several questions electronically within an assessment engine (MASS). The purpose of the MASS is to manage exams from different courses and the corresponding collection of questions for each exam. When each student takes the exam, he or she logs into the MASS and answers all questions. The MASS in turn stores all the answers in a database. Some input may be text (e.g. essays), others may be multiple choice answers which are easily handled by the MASS. Other input could depend on external services which have to be accessed by some sort of remote web service while the results of the students’ performance are being automatically returned. In our example, a student must solve a mathematical problem using ISAC (a graphical user-interface connected to a mathematics engine). The MASS notifies ISAC, the student completes his or her task and afterwards the MASS retrieves the results. Upon completion of the exam, the professor corrects the results and publishes them.

Figure 2.4a: The application environment
The basic idea is that the service provider we are looking at, ISAC, is called by some external entity, the MASS, in order to solve some mathematical problem and ISAC presents some sort of performance metrics back to the service provider.

From this scenario we are able to compile some use cases.

2.3.1 The Use Cases for Applying Web Technology

The use cases can be defined from at least two perspectives. From the point of view of the MASS or the service provider, ISAC. For the sake of completeness the use cases shall be mentioned from both perspectives. However, for the implementation only those parts of the use case will be regarded in which changes have to be made within the application code of ISAC.

**Prerequisites:**

The MASS already exists. A professor can create an exam for his class. He can also create questions for each exam. He may be able to define certain types of answers he would expect, like text, multiple choice, or others. In this master thesis we are only interested in those type of answers which the professor links to the service provider and to which he or she expects the performance result upon completion by the student. Thus the MASS should have some configuration of which URL it should contact in order to call ISAC and which URL it should contact to retrieve information from ISAC, like the performance result.

The implementation of these prerequisites is beyond the scope of this thesis. The following use cases (UC) can be created from the principle scenario:

**UC 1:** As stated before, within the MASS the professor of a class wants to create an exam. For each exam he can add questions and types of answers.
One type is where the MASS calls ISAC via a URL to complete the question. ISAC needs to start correctly if a user in the MASS clicks this link. Therefore, this URL needs to be specified. It should contain some authentication method and information of the user (e.g. user ID), which will be required in UC 3. It also contains some information to tell ISAC it should start the ‘assessment’ mode. Also, the name or ID of the exercise the user (student) must complete needs to be part of the URL. Thus a way to start ISAC via this URL is to be implemented.

**UC 2:** The student of the course has to complete an exercise within ISAC. The progress of each student needs to be logged for later retrieval.

**UC 3:** The manager of a course wants to see the results of how the student performed in the test, after the student completed it on ISAC. The MASS uses the user ID from UC 1 to retrieve the personalized information.

### 2.3.2 Evaluation with respect to the Use Cases

Most decisions concerning which technologies are to be used for implementing these use cases, were made during the evaluation of the architecture and the currently used technologies of the already existing software (MASS, ISAC). The result of the decision process can be seen in § 3 Architectural Design.

However we list here some challenges that must be overcome only from contemplating the use cases:

**Challenges from UC 1:** One major challenge is that the MASS is a web based application running in a browser on one computer which wants to start a desktop application on a different computer by clicking on a link which at this time can only be started per command line. The following question presents itself: can some command line applications from within a web browser be started? Additionally, it has to be taken into account that the application to be started (ISAC) is itself a client-server application (both the same binary file). It is desired that only the client part of ISAC is started by the MASS, which in turn connects to the already running the server part of ISAC. Once this has been solved, it must be made certain that the target
application (ISAC) does not just start statically every time in the same way; an ISAC process requires too many resources for multiple instantiation. Depending on which user is logged into the MASS and what exercise he or she wants to solve, the ISAC client needs to log in with the right user automatically, start itself in assessment mode and load the desired exercise. How can this be accomplished by just clicking on a link in the MASS?

In § 2 some frameworks are discussed as a basis to decide which approach is best suited to solve these challenges:

**JNLP**, as described in § 2.2.3, can start a remote application written in Java. Thus the ISAC front-end can be started using this technology. But how can it be started in such a way that it finds the ISAC server? Specific features of JNLP allow this question to be tackled.

**Challenges from UC 2**: Figure 3.3a shows, that a MASS and ISAC are completely separate applications. As soon as both are running (UC 1), UC 2 raises the question: What sort of data is needed from ISAC for evaluation within a MASS?

**Challenges from UC 3**: As soon as an exam (UC 2) has been finished, UC 3 presents the following question: How does MASS know from which ISAC instances (i.e. from which student) to collect which data (from a certain exam) for evaluation? Does this mean that the MASS server requires information to be retrieved from the ISAC server which has no web service capabilities? Should ISAC initiate the transfer of results to the MASS or should the MASS request them?

Other questions which must be explored include the following: How can a student be prevented from cheating by opening another session in ‘non assessment’ mode and using answers found there? What happens with a user’s previously opened sessions when an assessment starts? Who manages beginning and end times of an assessment (MASS or ISAC)? Is there any need of authentication? If so, which system is in charge of the user management?

Many of these questions are answered in the following chapters, while others
are simply raised and briefly discussed, and suggested for further exploration in other theses.
3. Architectural Design

Looking at the use cases from § 2.3.1 it first appeared to be straight forward as to what needed to be accomplished and which approach should best be used. After many difficulties had been overcome (see § 4) the more important analysis could be completed, namely that of the principle architecture of the applications that are to work together. Additionally, it is helpful to analyse every step of the user-application-application-user lifecycle in detail. Understanding these architectures and the lifecycle in the following chapters will lead to the final decisions to be made in § 3.4 and thus to a simple and general web interface as desired in § 1.

3.1 Architecture of MASS’s Web Service

The MASS we are looking at is web based (see use cases, § 2.3.1), therefore it is assumed that it can be accessed via a modern web browser and a URL.

On the one hand the MASS could be created with a simple web server that displays a couple of HTML files. On the other hand a more sophisticated web application could be used for running on an application server which could also be clustered and works with a database.
Two basic cases might be differentiated. In both cases a user visits the MASS’ domain via a web browser. Now, in the simpler case a web server simply looks in its local file system and delivers an HTML page directly back to the browser, which displays its content like Links to the ISACs exercises. The tasks regarding the MASS’ functionality are very basic in this case, because the web server may only display static HTML page, nothing more. In the more complex case, an application server may fulfill very complicated functional, analytical, navigational tasks in order to direct a user through some exercises and afterwards provide the course manager with analytical results. The dynamic state and functionality is managed via a server application (e.g. written in PHP, Java, .NET,...) that is most likely connected to some database where the final and intermediate results may be stored.

3.2 ISAC’s Architecture as an Example

ISAC is an experimental prototype focusing on features provided by Theorem Proving (TP). The architecture of the prototype is as general and as simple as possible in order to keep the system open for further development. The prototype has four modules running in (at least) four different JVM virtual machines which have been identified as follows:

1. The WindowApplication (WA) is the graphical user interface, one for each of an arbitrary number of students, connected to the server.
2. The ObjectManager (OM) accomplishes session management in the server and holds the dialogue components.
3. The Bridge is a Java wrapper for Isabelle/ISAC, the mathematics-engine running in a separate process under SML, the standard meta language.
4. The KES tore is the module accessing data of knowledge (theories, problems, methods) and of examples. These data are represented in different formats for different purposes.

(1) is the main component of the ISAC client that may be opened several
times on different computers. (2)-(4) are components of the ISAC server.

The following figure shows these modules.

Figure 3.2a: Architecture of the ISAC client and server

The connections between the modules vary with the specific requirements. For instance, Java RMI is used as the simplest and fastest technology to connect most of the modules that may run on different machines.

Particularly interesting for the planned web interface is the connection between the ISAC client, i.e. the front-end (WA) and the server (which itself consists of three RMI servers - the OM, the KEStore, the Bridge and the modules in the background and where Isabelle/ISAC is an external theorem prover engine). As mentioned above, the optimal solution for a planned web interface would be a browser based front-end for ISAC. [Sch07] performed a feasibility study on such a solution; the study led to rejection of this technology for the following reasons:

The connection between WindowApplication (WA) and ObjectManager (OM) is predominantly determined by message-passing in both directions: The OM waits for messages from the WA; this is the typical kind of a connection used and RMI and some other protocols are built explicitly for that. However
HTTP is stateless and principally speaking unidirectional. Also the mathematics-engine might be assigned requests which take a long time to finish. It was found that a web based front-end may run into a timeout. The responses of the mathematics-engine are then passed to the WAs as independent messages. This entails making the connection bi-directional by doubling the RMI interface for the other direction, which would go beyond HTTP’s features. [Sch07] experimented with JavaServer Faces (JSF) and asynchronous JavaScript and XML (AJAX). The results from [Sch07] showed that the system was too weak to cope with the complexity of the interaction required of an interactive mathematics assistant like ISAC.

The new solution presented in this master thesis is to use the current ISAC Java client as a Java Web Start application and extend the RMI interface between WA and OM to allow for a session to be started with autologin for a user in assessment mode and a given exercise.

### 3.3 User - Application Lifecyle

From § 3.1 and § 3.2 we can see what applications we are dealing with on both sides. Now we shall have a detailed look at each particular step that is involved in the whole user - application lifecycle. We need this later on in order to decide on which parts are to be newly developed or extended:
1. The MASS shows an URL within the browser as part of an assessment. The user (e.g., a student) clicks on the Link since he wants to start the ISAC client to finish the exercise there.

2. The user’s browser now refers to a different domain.

3. There an application server parses the Link for the authentication, user id, assessment mode and exercise information and returns a JNLP file to the user’s browser.

4. The browser then downloads the JNLP file and determines from the MIME type (see § 2.2.3) of this file that the browser should open the `javaws` application, taking the JNLP file as an argument.
5. This application downloads and starts the current version of the ISAC Java client.

6. The ISAC client connects via RMI with the ISAC server. The user is automatically logged into the ISAC system and the exercise is started in the desired assessment mode.

7. Upon finishing the exercise, the user returns to the MASS browser site. Now the MASS server may download the progress information from the ISAC server using a web service.

As it can be seen the overall process is not trivial and involves several steps to complete successfully.

3.4 Decision for the particular web technologies

We now have principle knowledge of the architecture of the MASS and the ISAC client and server. Additionally, we know every single step in the process of connecting the both according to the use cases. Now it is possible to decide which technologies are appropriate to implement which parts of the system:

a. The ISAC client, which was only able to be started via command line, is to be made started via Java Web Start (JNLP), since JNLP provides the only sensible way to start a Java application on a remote domain from a web browser on a different domain.

b. We need to feed additional dynamic parameters (user authentication a.s.o, see step 3 above) to the ISAC client. The JNLP file can take arguments in its XML structure which will be passed to the java application. However the XML structure needs to be changed every time according to the URL, which could look like that:

   http://<<someURL>>/downloadJNLP?institution=someInstitution&user=someUserId&password=somePassword&mode=assessment&example=example1

   This means the GET request parameter need to be parsed into JNLP file dynamically. javaws propagates these parameters further on to the ISAC client.

   Thus an application server has to be used and a server application has to be developed to parse each URL for these GET parameters and
dynamically create a JNLP file. The application server has to run on a system that can be reached via URL by the MASS.

c. The ISAC application client and server needs to be modified to use the dynamic parameters correctly. From Fig 3.2a we see that the WindowApplication will receive them from the Java Web Start process, since it is started first. Here at least the RMI interface between the WM and the ObjectManager needs to be extended to pass them on to the server. Further code adjustments must be made to add the functionality.

d. For the web service which will return results to the MASS, we need to extend the server application developed in b. At the moment the central part to receive requests within the ISAC server is the ObjectManager. It is an RMI server. This would be the perfect place to let the application server create an RMI connection to the ISAC servers ObjectManager. Its interface would have to be extended for the new functionality added for the web service. The ISAC server functionality would have further to be extended to read the progress information for a user from its backend. However, it was decided to directly query the ISAC’s database for the information needed instead, the reason being, that it is currently uncertain how the interface would evolve. This information will be transferred back to the OM then via RMI to the application server and to the MASS server.

For details, how this had been accomplished see § 4. As discussed in § 2 there are several web application frameworks that can be used. In this master thesis it was decided to use Struts + JSP as an MVC framework. Any other framework would have done as well (like JSF, as discussed in § 2). This decision was made since the author knew both technologies well enough to more efficiently implement the features using Struts.

For the web service framework and protocol it was decided to use REST rather then WSDL + SOAP since the web service to implement was very simple, in fact it was too simple to use a complex protocol like SOAP. It has been analysed that a REST services is a lot easier to implement on the ISAC
side and on the MASS side, where already existing JavaScript browser API’s or application server API’s would be used.

Thus one Struts Action + JSP page will return the dynamically created JNLP file. Another one can be used for the REST service.

To integrate the start of ISAC on the MASS side, from the scenario described in § 3.3, a simple HTML URL needs to be inserted in the HTML page. In the simplest case the code for this could look something like this:

```html
<a href="http://<<someURL>/downloadJNLP?institution=someInstitution&user=someUserId&password=somePassword&mode=assessment&example=example1">Example 1</a>
```

In the following chapter we will show how each use case has been implemented in detail.
4. Detailed Design and Implementation for ISAC’s Web Interface

In § 2 the three principal use cases were defined. From these use cases basic architectural decisions have been made. Now we look how this decisions have been implemented in detail. In the seminar-project, upon which this thesis builds, we only looked at the adaptations that needed to be made at the ISAC client and server. This master thesis also shows how the implementation process started using user stories and how some of those were implemented. It also looks into the implementation of the new server application and the installation of the same and it also goes into more detail of the development process and also how ISAC is to be installed and the difficulties that were faced and more.

4.1 The Implementation process starts with User Stories

From these use cases that were defined in § 2, basic architectural decisions have been made.

Now for the detailed design and implementation each use case is to be subdivided into single user stories.

“User stories are used with agile software development methodologies as the basis for defining the functions a business system must provide, and to facilitate requirements management.”

Each user story fulfills at least one or part of one business value. After completing one user story the whole system is still fully functional (nothing should be broken), but an additional business value has been gained. Working with user stories in this way keeps the progress of the whole project highly visible and also keeps one user story small and simple. However, some user stories may depend on the completion of others, in order to bring full functionality.

Below there are user stories (US 1 - US 10) for the whole process of setting up ISAC and in a second step to implement the features necessary. Each user story is also accompanied by simple acceptance criterions which “specify
scenarios to test when a user story has been correctly implemented.

### 4.2 Setting up ISAC

Before it was possible to analyse and/or implement new functionality within ISAC, the installation of the ISAC client-server application proved to be a quite complex task. The main reasons were that before any new implementation was to be made, the code was to be transferred from an older versioning system (CVS) to a newer one (Mercurial). Then, since development of ISAC stalled for some time, an older version of its mathematical backend (Isabelle) needed to be upgraded to a newer version and also to a newer operating system. At the same time the Java part of the project (ISAC) had to be made ready to run on a different Development Environment, Netbeans instead of Eclipse. Then a lot of configurations needed to be figured out in order to set up the connections between the many components (RMI, data base, XML-RPC, etc).

This chapter outlines the tasks that had to be done first and lists them in user stories. A complete installation consists of the following steps which will be described in detail in the user stories US 1 - US 4:

- Install Virtualbox
- Create new Isabelle Instance from scratch or import an existing application from Isabelle2002neu.vdi file.
- Create a Port forwarding from host port 2222 to the Isabelle guests port 22.
- Download the source code from the ISAC homepage and import it into the Netbeans IDE.
- Install an ssh client.
- Create an ssh key and copy it within Isabelle’s virtualbox
- Test the ssh connection
- Create and copy the .java.policy file to the <home> directory.
- Correct paths in the start configuration of ISAC and the .properties files
- Configure the application server for the web application
- Start the rmiregistry
- Compile the source code for ISAC and the web-application
- Start the Bridge, KEstore, ObjectManager, WindowApplication (in this order)
- Start the application server

**US 1: As a developer I want to use Mercurial instead of CVS as a versioning system. The old CVS repository will not be used any more, but the versioning history should be kept, if possible**

*Implementation:* § 4.4

It needs to be researched how to export the CVS repository and import it into Mercurial without loosing the history. Also, mercurial should be accessible via web. And, also it needs to be understood by the developer how Mercurial is to be used during development.

*Acceptance criteria:*
- The mercurial repository represents the current CVS content, optionally with the history.
- It is possible to create a local mercurial repository from the master repository
- Changes may be downloaded or committed if the user has the proper authorization to do so.

**US 2: As a developer I want to develop on a Windows based computer. Since, at this point, the Isabelle mathematics engine only runs within Linux, a virtual application should be created, to which ISAC (from the Windows host computer) can connect via network.**

*Implementation:* § 4.5

Create a virtual machine with Ubuntu installed (eg. using Virtualbox). Install Isabelle within the virtual machine. Make it possible to connect and start Isabelle via ssh. Export the ports to the environment, so that it can be communicated with Isabelle from outside the virtual machine (from ISAC)

*Acceptance criteria:*
- A remote connection from outside the virtualbox can be made to Isabelle via ssh and command line call to successfully start it
US 3: As a developer I want to develop the ISAC code base using the Ant based Netbeans project system instead of the Eclipse based project system.

*Implementation: § 4.6*

The ISAC codebase should be running within Netbeans. All dependent libraries should be linked into the project. It should be compiled using Java JDK 1.6. The Netbeans Mercurial plugin should be used to commit and update the code base.

*Acceptance criteria:*

- ISAC can be compiled using JDK 1.6 within Netbeans without any dependency errors.
- Mercurial can be used within Netbeans to commit or update code.

US 4: As a developer I need to setup ISAC correctly in order to communicate with Isabelle and start without errors.

*Implementation: § 4.7*

Following property files need to be changed: *BridgeMain.properties, KEStore.properties, ObjectManager.properties, WindowApplication.properties, bridge.ini*. The command file *START_ISABELLE.cmd* to remotely call Isabelle also needs adaption. A `*.java.policy` file for the Java JDK needs to be created to correctly run the RMI service. Also a MySQL database for the UserLogger needs to be installed and configured.

*Acceptance criteria:*

- Run the *rmiregistry.exe*
- Running *BridgeMain* correctly connects and starts Isabelle.
- Running *KEStore* correctly starts and connects via RMI.
- Running *ObjectManager* correctly starts and connects via RMI.
- Running *WindowApplication* correctly starts the GUI and connects via RMI.
- Authentication should work correctly.
• No RMIException should be thrown.
• It can be connected to the MySQL server.

### 4.3 User Stories for the new functionality

At this point it should be possible to start ISAC and the backend (see § 4.8). In order to implement the use cases UC 1 - UC 3 (see § 2.3.1) we first subdivide them as much as possible into as simple as possible user stories.

**US 5: As a user, working within a MASS in a web browser, I want to click on a HTTP Link, which should call ISAC. This Link contains GET request parameter that ISAC needs to receive.**

Satisfies: user story UC 1 (§ 2.3.1).
Implementation: § 4.9

The JNLP technology does not support dynamic parameters, like HTTP GET parameters to be passed on to the Java program to start. For example the MASS would need to call ISAC using different parameters, like different students (session id’s) or different test exercises. However JNLP does support parameters within its JNLP descriptor. These parameters are then passed to the java program that is also configured within this JNLP descriptor.

• It needs to be analysed and implemented what additional software package can be used in order to create JNLP descriptors dynamically.
• Implement the new functionality within ISAC or as an extra package.
• It is to be defined under which URL the dynamically created JNLP descriptor can be accessed.
• The **HTTP GET request parameters** needs to be defined. Following parameters are needed:
  ○ authentication of the MASS: *username, password*
  ○ *id* of the user of the MASS
  ○ mode to operate: *assessment mode, normal mode*
  ○ the *exercise id* that the student needs to complete.
• These parameters need to be parsed into the JNLP descriptor.
Acceptance criteria:

- The ISAC application can be started by clicking on an URL
- The request parameter in the URL, that contains
  - the authentication of the MASS
  - id of the user of the MASS
  - mode to operate
  - the exercise id

need to be parsed into the JNLP descriptor so that when the ISAC application is started, the parameters can be used within ISAC.

**US 6: As the ISAC server (WindowApplication) I need to create a new user session containing the parameters from the HTTP(JNLP) URL**

Satisfies: user story UC 1 (§ 2.3.1).
Implementation: § 4.10

At the current state ISAC contains a simple user/login management. When ISAC starts, a login window is shown, where the user types in his credentials. For each user a specific environment is loaded. This environment defines how ISAC should act and what it should show and what can be accessed within ISAC. For each user who logs in, a new user session is created within ISAC, which lasts until the user closes the ISAC main window.

This simple user management needs to be extended. A MASS maintains a list of exams for courses. Each exam is taken by a group of students. Because each student of one course sees the same questions also ISAC needs to set up the same environment on a course basis.

Also ISAC does not want to create environment definitions for every student, since that would mean that ISAC must store the credentials of every student. Instead it is sufficient to create an environment for a group of users, meaning eg institutions and courses. This makes perfect sense, since the same environment settings apply for the whole group of students who take the exam.

However ISAC still needs to record the students progress (see user stories
further down), therefore ISAC needs following information:

- the MASS authentication information for a course or a group of students
- it needs to store the user (in order to record the progress for each student) to
  1. automatically log the user in and skip the login window
  2. store this information for each user session that is created via JNLP (see US 5).

Other information that needs to be stored in the user session are:

- the assessment mode
- and the example id that the student has to accomplish
(see US 5 for detailed specification)

**Acceptance criteria:**

- If ISAC is started via JNLP, it will automatically log the user in, without showing the login window
- ISAC creates a new user session using the MASS name, the students user id of the MASS and password, the ‘assessment’ mode and the exercise id.
- Load settings common to a group, rather than specific to a user.

**US 7: As a user, working within a MASS in a web browser, if I click on a HTTP Link I want the ISAC client to open the example within the currently open ISAC client.**

Satisfies: user story UC 1 (§ 2.3.1).
Implementation: § 4.11

Via US 5 and US 6 it is possible to start the ISAC client on the clients computer, passing all necessary parameters to it. In this user story following should be handled as well:

- If the user clicked on an HTTP link on the MASS to open ISAC for the first time (where no ISAC client is open for this user yet) then the JNLP file should be downloaded and ISAC should be started as specified
before. Additionally the example that has been passed by URL should be opened in a new worksheet.

- If the user clicked on a HTTP link on the MASS to open ISAC for any subsequent time (an ISAC client is already open for this user), then no JNLP download should happen, but the example that is been passed by URL should be opened in the existing ISAC client.

**Prerequisites:**
- User is authenticated
- JNLP parameter for 'example' contains a valid id known to ISAC

**Acceptance criteria:**
- If the user clicks on an HTTP link to open ISAC for the first time, then ISAC is being started (as before) and the example from the URL is being opened
- If the user clicks on the HTTP link the second time with the ISAC client currently open for this user, then the example opens in the current ISAC client.

**US 8: As a user I need the session on the ISAC server to be started in 'assessment' mode.**
Satisfies: user story UC 1 (§ 2.3.1).

This user story only applies if the JNLP parameter 'mode' is 'assessment'. This mode enforces certain settings within ISAC. For example, the 'AUTO' button disappears, certain knowledge items are invisible, etc. If this is the case and the user has been logged in, then ISAC should immediately show the example that has been configured via the JNLP parameter 'example'. The student can start the exercise right away.

If a user session for a user has been created in 'assessment' mode then the user can not log in again in any different mode.

**Prerequisites:**
• User is authenticated
• JNLP parameter for ‘mode’ = ‘assessment’
• JNLP parameter for ‘example’ contains a valid ID known to ISAC

Acceptance criteria:
• The Assessment mode for the student starts correctly
• A student can start the exercise
• In Assessment mode another open ISAC session that is not in ‘Assessment’ mode should be closed.

**US 9:** As the ISAC server I need all progress that the user makes within one session to be logged.
Satisfies: user story UC 2 (§ 2.3.1).

ISAC already tracks user interactivity. It has to be made sure that the user information of the user that logged in via a MASS is correctly logged together with all the information of what the user did within ISAC.

Acceptance criteria:
• The user information of the user that logged in via a MASS is stored together with the information what progress the user made.

**US 10:** As a professor of a previously, on a MASS, created course I want to retrieve the progress information for the students from the ISAC server.
Satisfies: user story UC 3 (§ 2.3.1).

After the student finished his exercise within ISAC, the MASS must be able to retrieve the progress information from ISAC. ISAC should provide some web service interface to retrieve this information, which needs three parameters:
• the credentials of the MASS for authentication:
  ○ the ID of the MASS (e.g. institution name)
and the password

- the user ID of the student of the MASS for who ISAC stored the progress information.

The web service should be accessible via a URL. This URL is dependent on where ISAC has been deployed. The name of the interface should be:
/service/progress

Thus the URL would be something like:

http://<<someURL>>/service/progress

It is to be evaluated as part of this user story which underlying technology is to be used for this web service.

Acceptance criteria:

- ISAC needs to provide a web service interface to access the progress information of a user from an MASS.
- The web interface can be reached under the URL

  http://<<someURL>>/service/progress

- The URL needs to contain the credentials for the MASS and the user ID of the student in order to retrieve the progress information for the student.

4.4 Converting and working with Mercurial

A place where CVS repository converters can be found is at mercurials homepage. The currently most reliable converter seems to be cvs2hg which is a branch of the cvs2svn converter. Once it has been set up according to official instructions then only one command is needed:

   ./cvs2hg --hgrepos myproject.hg /path/to/cvs/myproject

Now the repository is converted to the ./myproject.hg folder.

The next step is to make this repository accessible to the internet. The simple way is using mercurials lightweight http server, via the command

   hg serve
Now it can be connected to ‘http://localhost:8000’. However only one user can connect to this port and it lacks stability and authentication mechanisms. A more reliable way is using a proper web server, using a special script that allows the web server to serve as a proxy to the mercurial repository. With this multiple developer can connect to the main repository at the same time and it is also possible to restrict read and write permissions on a per-user basis, using different authentication mechanisms (e.g. username/password or certificate based) or allow for encryption of the data, e.g. using ssh.

Mercurial has the big advantage to other versioning systems, like CVS or SVN, that every developer always creates a clone of the current main repository on his local machine and now is able to work disconnected from the internet and can commit changes to his local repository as if he is actually connected to the main repository. Every developer can commit to his or her local repository first, and when ready commit the whole version history, since the last synchronisation to the main repository, later.

There are 4 main operations:

- Creating a clone from a repository to the local machine
  
  `hg clone <<URL of remote repo>> <<local repo name>>`

- Committing a change from the code base (this is the developers source code he is working on) to the local repository
  
  `hg ci`

- Pulling changes from the main repository to the local repository. This will bring changes that have been made to the main repository by others to the local repository.
  
  `hg pull`

- Updating changes from the local repository to the code base. ‘Pulling’ only affected the local repository so far. Doing an update will now actually change the developer source code.
  
  `hg up`

- Pushing all changes that are in the local repository to the main
repository. After the developer changed the source code and may or may not have committed several changes it to the local repository, ‘pushing’ will bring all those changes to the main repository.

\texttt{hg push}

This step can only be done if changes have been ‘pulled’ and ‘updated’ from the main repository in the first place.

4.5 Create a virtual application for Isabelle, the Theorem Prover

Since it was needed to develop ISAC on Windows and Isabelle only ran on Linux it was necessary to create a virtual environment running Linux, where Isabelle could be installed.

The goal of the following steps is that it is possible to simply connect via \texttt{ssh} to Isabelle and start it via the ssh tunnel.

1. Create a Virtualbox with Ubuntu to install Isabelle.
   Download and install the Virtualbox Software\textsuperscript{51}. Then download the .iso file for Ubuntu and install Ubuntu\textsuperscript{52} within Virtualbox. Here, Ubuntu Desktop 10.04 was used.

2. Download Isabelle 2002.
   First we need to download the .tgz file from the Isabelle Homepage\textsuperscript{53}

3. Unpack \texttt{Isabelle2002-isac-NOheap.tgz} to /usr/local/

4. Download PolyML 4.1.4 from the homepage\textsuperscript{54} and unpack the \texttt{polyml-4.1.4.tgz} to /usr/local.

5. It was tried find dependency errors for PolyML by using
   \texttt{ldd poly}
   in /usr/local/polyml-4.1.4. \textit{It appeared that the file libXm.so.3 did not exist.} We found out via the internet\textsuperscript{55} that we needed to install another package using
   \texttt{sudo apt-get install libmotif3}
   Calling \texttt{ldd poly} again showed that no dependency was missing.

6. Now calling
   \texttt{/usr/local/poly-4.1.4/x86-linux/poly}
starts the interpreter successfully showing

Copyright (c) 2002-5 CUTS and contributors.
Running with heap parameters
(h=10240K,ib=2048K,ip=100%,mb=6144K,mp=20%)
WARNING:./ML_dbase: Write permission denied.
WARNING:./ML_dbase: Opened for reading only.
Mapping ./ML_dbase
Poly/ML 4.1.3 Release
>

7. Press CTRL-D to exit.

8. Now it is needed to change the settings for Isabelle, such that it finds
the polyml interpreter. Open the file

    /usr/local/Isabelle2002/etc/settings

in an editor and change the contents after the first `if else fi` block
to

    ML_SYSTEM=polyml-4.1.4
    ML_HOME=/usr/local/polyml-4.1.4/x86-linux
    ML_DBASE=/usr/local/polyml-4.1.4/unknown-platform/ML_dbase
    ML_OPTIONS="-h 15000"
    ML_PLATFORM=x86-linux

9. In the next step we do a

    ./build Pure

to build Isabelle, which shows (abbreviated)

    ********************************************
    * Welcome to Isabelle build *
    ********************************************

    ....

    Press RETURN to compilation of
    Pure

and after pressing return

    Started at Tue Dec 21 11:15:40 CET 2010 (polyml-4.1.4_x86-linux on
tom-VirtualBox)
    Building Pure ...
    Finished Pure (0:00:25 elapsed time)
    Finished at Tue Dec 21 11:16:07 CET 2010
    0:00:27 total elapsed time
Doing a

    find -name Pure in /usr/local/Isabelle2002/

returns a

    ./heaps/polyml_unknown-platform/log/Pure
    ./heaps/polyml-4.1.4/x86-linux/Pure
    ./src/Pure

showing that it compiled correctly.

10. Now it is needed to compile HOL, using

    ./build -m HOL-Real

This will give a similar response as before.

Doing a

    ls heaps/polyml-4.1.4/x86-linux/

will show that several directories were created

    HOL HOL-Real log Pure

11. It is also needed to create HOL-Real from HOL-Real-Isac since ISAC uses this. But these heaps binaries need to exist first. So they are to be copied first

    cp heaps/polyml-4.1.4/x86-linux/HOL-Real
    heaps/polyml-4.1.4/x86-linux/HOL-Real-Isac

12. Also ISAC needs the sml data, so they must be unpacked

    cd /usr/local
    sudo tar xvfz sml.tgz
    cd /usr/local/polyml-4.1.4/x86-linux/
    chmod u+w
    /usr/local/Isabelle2002/heaps/polyml-4.1.4/x86-linux/HOL-Real-Isac

Now calling

    ./poly
    /usr/local/Isabelle2002/heaps/polyml-4.1.4/x86-linux/HOL-Real-Isac

will return

    Copyright (c) 2002-5 CUN and contributors.
    Running with heap parameters
Now type this into the prompt to see if the interpreter works

cd “/usr/local/sml”;
val it = () : unit

> use “RCODE-root.sml”

... This will show that a lot of calculations are being done.
After poly finished, quit poly with CTRL-D and the results will be written into

... /HOL-Real-Isac

13. The installation of ISAC is finished now.

14. For conveniency a symbolic link for poly should be created

cd /usr/local

sudo ln -s /usr/local/polyml-4.1.4 poly

cd /usr/local/bin

Create a file called poly in this directory, containing

cd /usr/local/poly/x86-linux

exec ./poly $*

Make it executable via

chmod a+x poly

Now it is possible to call poly from every directory and it will be started correctly. Trying it out

poly /usr/local/Isabelle2002/heaps/polyml-4.1.4/x86-linux/HOL-Real-Isac

will start the kernel with the mathematic engine and the knowledge base for ISAC. Within the > you can try

thy;

which will show all theories within the knowledge base
15. Install a ssh server. This is needed in order to connect to the Virtualbox installation via ssh. Type

```bash
sudo apt-get install openssh-server
```

to install the ssh daemon on the machine

16. In order to connect from the windows host via ssh to the ssh server within the linux guest within the Virtualbox we need to configure the Virtualbox installation to do a port forwarding, eg from port 2222 on windows to port 22 within linux. This can be done by opening Virtualbox, open the ‘Settings’ dialog of the Linux installation, goto ‘Networks’ and ‘Adapter 1’. Make sure ‘NAT’ is selected. Then within ‘Advanced’ ‘Port Forwarding’ is to be opened. Here, ‘Insert a new rule’ and type

Rule name:
Protocol: ssh2222
Host IP: 127.0.0.1
Host Port: 2222
Guest IP: 127.0.0.1
Guest Port: 22

Now it is possible to connect via ssh to the Linux installation within Virtualbox

17. Making ssh work between windows and the Linux virtualbox. Since the port-forwarding is working now, a ssh client is need on the windows side.

a. For this, download cygwin and install it into the `c:\cygwin` (or any other folder that does not contain spaces in the path). From the cygwin package manager only select openssh as ssh client. Also put `c:\cygwin\bin` into the Windows PATH environment variable.

b. Start the cygwin bash using `cygwin.bat`.

c. Create a ssh key using: (Make sure no passphrase is entered)

```bash
$ ssh-keygen -t rsa
```

Generating public/private rsa key pair.
Enter file in which to save the key (/home/a/.ssh/id_rsa):
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tom/.ssh/id_rsa.
Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:

A key has been generated with in the
/cygwin/home/<username>/ssh folder

Now use ssh to create a connection as user <username> (eg ‘tom’) on localhost on port 2222, which is the Isabelle virtual box.
(The directory may already exist, which is fine):
$ ssh -p 2222 tom@localhost mkdir -p .ssh

Finally append the new public key to authorized_keys file in the virtualbox and enter the for the password <username> one last time:
$ cat .ssh/id_rsa.pub | ssh -p 2222 tom@localhost 'cat >> .ssh/authorized_keys'

From now on it can be logged into Isabelle’s virtualbox via ssh as <username> from your host machine without password. Since cygwin was put to the windows PATH variable, it is possible to start ssh from a windows command shell (cmd.exe), eg by typing
ssh -p 2222 tom@localhost

If this error appears: "Permissions 0660 for '/home/Thomas/.ssh/id_rsa' are too open. Change group to Benutzer on .ssh folder", then do
chgrp <username> ~/.ssh
chgrp <username> ~/.ssh/*
chmod 600 ~/.ssh/id_rsa

4.6 Making ISAC work as a Netbeans project

This step is actually quite simple:

1. Download the Java SE Development 6 (JDK 6) edition from the Java homepage\textsuperscript{57} (not the Java SE Runtime)
2. Download Netbeans website\textsuperscript{58} (Use the ‘Java EE’ edition)
3. Within Netbeans download the source by creating a Mercurial (hg)
4.7 Configuring ISAC and Netbeans

ISAC consists of several different components that share the same source code library, but can in fact be started separately. Here it is described how to configure them before it is possible to start them:

4.7.1. Install MySql via xampp for the UserLogger table

xampp is a collection of conveniently packaged server products for multiple operating systems. It consists of an Apache Web-, MySQL-, PHP- and Perl-server and other components. Here only MySQL and the web server is needed to administer the database. MySQL is used by ISAC to store data from the UserLogger.

1. Download the installation package.
2. Install it into c:\xampp. Don’t use the ‘Program Files’ folder, since xampp does not have sufficient write privileges in this folder.
3. Start mysql server from xampp control panel
4. Change the directory to c:\xampp\mysql\bin and type
   mysql -uroot
   You will be prompted with
   mysql>
5. Type
   source
<<root_dir>>/develop/isac/isac_clone/isac_java/src/sql/User
Logger.sql
to add a new table database in the mysql server. You will be prompted with
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 1 row affected (0.00 sec)
...
Records: 0  Duplicates: 0  Warnings: 0

6. Type exit to end the mysql session

4.7.2. Configure ISAC to connect to Isabelle via ssh client

1. Create a properties/ folder in
<<root_dir>>/develop/isac/isac_clone/.

2. Copy all files from folder
<<isac_java>>/src/java/properties-templates.windows/ to
<<root_dir>>/develop/isac/isac_clone/properties/
This folder (<<properties_dir>>) contains all important properties to
the different modules of ISAC.

3. Install an openssh client60 Try it out by opening a command line
window and type
ssh
It should show instructions how to use it.

4. Now start the *Isabelle* Virtual machine we created in § 4.5 and try to
connect to it via ssh:
ssh localhost -p 2222 -l tom
Exit again via typing exit.

5. In order to let the Java kernel connect to Isabelle we need to configure
how they should talk to each other. We need to create a start file,
name it <<properties_dir>>/START_ISABELLE_2002.bat and add
this line to it
ssh localhost -p 2222 -l tom exec poly
/usr/local/Isabelle2002/heaps/polyml-4.1.4/x86-linux/HOL-Real-Isac
This should connect via ssh to the ssh server in the Isabelle virtualbox and start the Isabelle processor. Exit out of it and continue.

4.7.3. Configure ISAC

1. Change all settings in the <<properties_dir>> folder as follows and replace all <<...>> placeholder with the actual file path.

   **For the BridgeMain.properties file change:**

   INI_PATH=<<properties_dir>>/bridge.ini
   DTD_PATH=<<isac_java>>/src/xml/isac.dtd

   **For the KEStore.properties file:**

   GENHTML_PATH=<<isac_java>>/src/java/isac/util/genhtml/
   XML_DATA_PATH=<<root_dir>>/develop/isac_clone/isac_repos/xmldata/
   GENHTML_DESTINATION_PATH=<<root_dir>>/develop/isac/www/kbase/
   LOG4J_PROPERTIES=<<isac_java>>/src/java/log4j.properties

   **For the ObjectManager.properties file:**

   WEB_PATH = file://<<root_dir>>/develop/isac/www/kbase/
   SESSION_MANAGER_USER_PATH = <<root_dir>>/develop/isac_clone/isac_repos/xmldata/users/
   LOGGER_DATABASE_ENABLED = true
   LOGGER_DATABASE_PATH = localhost
   LOGGER_DATABASE_PORT = 3306
   LOGGER_DATABASE_NAME = userlogger
   LOGGER_DATABASE_USER = isac
   LOGGER_DATABASE_PASSWORD = isac

   **For the bridge.ini file:**

   path=<<isac_java>>/build/classes/
   kernelExec=<<properties_dir>>/START_ISABELLE_2002.cmd

   The WindowApplication.properties can be left unchanged.

2. Create a folder www/ into the <<root_dir>>/develop/isac/ folder and unpack the kbase.tgz file into it.
3. Create a .java.policy file in the <home> directory (eg.
This is needed to allow connections via RMI to the rmiregistry. Add these lines:

```java
grant {
    permission java.net.SocketPermission "*:1024-65535", "connect,accept,listen,resolve";
    permission java.net.SocketPermission "*:80", "connect";
    permission java.lang.RuntimePermission "setFactory";
    permission java.lang.RuntimePermission "setIO";
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.util.PropertyPermission "*", "read,write";
    permission java.security.AllPermission;
};
grant {
    permission java.io.FilePermission "<ALL FILES>"", "read, write, delete, execute", signedBy "ISAC";
    permission java.io.FilePermission "C:\Benutzer\tomzi\develop\isac\isac_clone\isac_java\build\classes \", "read, write, execute";
};
```

If there are problems with `java.security.AccessControlException` when starting ISAC later, then Java cannot find the policy file. Try adding a line to the

```java
<<JDK_6_install_folder>>/jre/lib/security/java.security file:
policy.url.3=file:C:/Documents and Settings/<<your username>>/.java.policy
```

4. Create the directory `/sml/BridgeLog/` in the `<<isac_java>>/build/` directory.

5. Create a Run Configuration for `BridgeMain`. Open Netbeans, click on `Run/Set Project Configuration/Customize/` and enter for `Main-Class`: `isac.bridge.BridgeMain`  
   `Arguments`: `<<properties_dir>>\BridgeMain.properties`  
   `VM-Options`: `-Djava.rmi.server.hostname=127.0.0.1`  
   `-Djava.rmi.server.codebase='file:/<<isac_java>>\build\classes'`  
   `file:/<<isac_java>>\build\test\classes\`  
   `file:/<<isac_java>>\lib\log4j-1.2.11.jar'`  

6. Create a Run Configuration for `KEStore`:  
   `Main-Class`: `isac.ks tore.KEStore`  
   `Arguments`: `<<isac_java>>\isac_java\src\java\properties\KEStore.properties`  
   `VM-Options`: `(same as in Step 5)`

7. Create a Run Configuration for `ObjectManager`:  
   `Main-Class`: `isac.bridge.ObjectManager`
Arguments:
<<iasc_java>>/isac_java/src/java/properties/ObjectManager.properties

VM-Options: (same as in Step 5)

8. Create a Run Configuration for WindowApplication:
   Main-Class: isac.bridge.WindowApplication
   Arguments:
   file:/X:/<<iasc_java>>/isac_java/src/java/properties/WindowApplication.properties
   VM-Options: (same as in Step 5)

9. Create another Run Configuration for WindowApplication (name it eg WindowApplication -username:
   Main-Class: isac.bridge.WindowApplication
   Arguments:
   <<iasc_java>>/isac_java/src/java/properties/WindowApplication.properties -username ae/tomzi2 -password ae -mode assessment
   VM-Options: (same as in Step 5)

d. Configure the application server

The web application /isac-web has already been downloaded and a Netbeans project has been created from this source in § 4.6.

- To configure the database connection from the application server to the UserLogger of ISAC (see § 4.13 for details) the
  \isac-web\setup\glassfish-resources.xml file needs to be configured:

```
<property name="serverName" value="localhost"/>
<property name="portNumber" value="3306"/>
<property name="databaseName" value="userlogger"/>
<property name="User" value="isac"/>
<property name="Password" value="isac"/>
<property name="URL" value="jdbc:mysql://localhost:3306/userlogger?zeroDateTimeBehavior=convertToNull"/>
<property name="driverClass" value="com.mysql.jdbc.Driver"/>
```

It needs to be made certain that the properties for the jdbc-connection-pool are configured such that the application server connects to same database as ISAC does. The properties in ISAC can be found in the ObjectManager.properties file above.

- Now the /isac-java project can be compiled and the files
  <<iasc-java>>/dist/isac-java.jar
  <<properties_dir>>/WindowApplication.properties
need to be copied to the /isac-web/web/WEB-INF/resources/jnlp/ folder.
This is the file that will be downloaded via JNLP to the clients computer and from which the WindowApplication will be started, which in turn will download the WindowApplication.properties file. Make sure that the URL's configured in this WindowApplication.properties contain the URL of your public server, so that all the clients do not connect to localhost but to the correct server.

That's it, now it should be possible to 'Run' or 'Debug' the /isac-web project. Netbeans will compile the project, then configure Glassfish to deploy the isac-web.war file and copy the glassfish-resources.xml configuration to <<glassfishDir>>/domains/<<yourDomain>>/domain.xml file. Then it will start the Glassfish server.

This is sufficient for development, but if the installation needs to be deployed on the companies Glassfish (or any other J2EE compliant server) then the isac-web.war file needs to be copied manually and the contents of the glassfish-resources.xml needs also to be configured in the companies Glassfish's domain.xml file.

**4.8 Starting ISAC for the first time**
Everything should now be ready to start or debug and develop new features within Netbeans.
ISAC needs to be started in several steps:

1. First Isabelle needs to be running. Start the Isabelle Virtualbox we created earlier.
2. Start the RMI registry by opening a command line terminal and enter\(^\text{63}\):
   
   ```
   <<jdk_home>>/rmiregistry 1099
   
   ISAC is now able to register its RMI services to the registry.
   ```
3. Next, the three server instances of ISAC are to be started. Click within Netbeans on the isac-java project, then click on the 'Run
Configuration’ BridgeMain in the toolbar and select ‘Debug Project’ icon next to it. Continue by selecting the Run Configuration for KEStore next and click ‘Debug Project’. Do the same for Objectmanager and WindowApplication. This should open a window with the ISAC frontend running.

Now the isac-web server can be started via ‘Run’ or ‘Debug’. Per default it starts a browser with the URL http://localhost:8080/isac-web/index.htm which shows a couple of example links, like:

```
http://localhost:8080/isac-web/downloadJNLP.do?username=ae/tomzi&password=ae&mode=assessment&example=exp_IsacCore_Tests_1a
http://localhost:8080/isac-web/downloadJNLP.do?username=ae/tomzi&password=ae&mode=assessment&example=exp_IsacCore_Tests_1b
```

Clicking on one of those links would download a JNLP file and start the ISAC WindowApplication client (see § 4.10 for details and also on how the login process works)

**4.9 Dynamically generating the JNLP file - Creating a web application**

From § 3.3 and 3.4 it has been analysed that an application server needs to receive the HTTP GET request of the user from the MASS. The additional information encoded in the GET request is then used such that a JNLP XML file can be created dynamically by the web application to finally start the ISAC web application on the users computer.

First we define how the URL should look like. From US 5 we know that we need to encode a *username*, *password*, an assessment *mode* and the *example ID*. It is good practise to use similar or the same names of parameters and values related to the domain you are working in. Therefore we encode the HTTP GET request accordingly:

```
?username=someUserId&password=somePasswd&mode=assessment&example=example1
```

This is the *query* part of the URL. This part will be parsed into the JNLP file.
The complete URL as seen from the MASS perspective might look sth like this:

```
http://myhost.com:8080/isac/app/download?username=ae/tomzi&password=ae&mode=assessment&example=exp_IsacCore_Tests_1a
```

The external URL `http://myhost.com:8080/isac/app/download` is mapped by the companies web servers according to the internal server structure and configuration. If we start this web application within Netbeans (see § 4.6), the ‘internal’ URL will look like:

```
http://localhost:8080/isac-web/downloadJNLP.do?username=ae/tomzi&password=ae&mode=assessment&example=exp_IsacCore_Tests_1a
```

The first part of the URL path (here `/isac-web`) is part of the Glassfish configuration. All mappings to right side from this URL path are defined within the web application. For example, in the `web.xml` it is configured that all URL paths that end with `.do` are to be handled by `Struts`. As described in § 2.2.2 Struts’ main configuration is done in the `struts-config.xml`. Currently one URL path ‘downloadJNLP.do’ is mapped to the `Action` handler `WebAccessAction`. Struts automatically parses the `query` information of the rest of URL (username, password, ...) and saves it to the `WebAccessForm` (which is also configured in the `struts-config`). Right now the only thing that `WebAccessAction` needs to do is forward to the ‘SUCCESS’ page, which is served by the `/WEB-INF/jsp/jnlpDownload.jsp` file. Later (§ 4.11) this will be extended. The jsp file renders a XML, the JNLP file:

```
...<c:set var="codebase" value="<%=request.getScheme() + ":/"%>
  request.getServerName() + ":" + request.getServerPort() +
  request.getContextPath() + "/" %>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<jnlp codebase="${codebase}" href="" spec="1.0+">
  <information>
    <title>ISAC Webclient</title>
  </information>
  <resources>
    <j2se version="1.6"
      java-vm-args="-Djava.rmi.server.hostname=<%=request.getServerName()%>"
      href="http://java.sun.com/products/autodl/j2se"/>
    <jar href="resources/jnlp/isac-java.jar" main="true"/>
```
Here it can be seen which resources are being downloaded additionally. They are stored in the /resources/jnlp/ folder which contains all libraries and the WindowApplication.properties file. The WindowApplicationWebStart is being called as the main class and all parameters from the original URL are being passed to it, via EL commands (eg ${param.username}). Also the client Java virtual machine is being configured by the JNLP file such that it is able to connect to the ISAC server via ISAC’s RMI server (note the ‘-Djava.rmi.server.hostname=...’ argument).

Within the ISAC’s main class WindowApplicationWebStart, one of the first things that happens is to parse the arguments from above:

```java
public class WindowApplicationWebStart extends WindowApplication {
    public static void main(String[] args) throws RemoteException {
        Options options = createPossibleCommandLineOptions();
        CommandLine line = parseCommandLineOptions(options, args);
        WindowApplication.CommandLineValues commandLineValues = processCommandLineOptions(line, options);
        WindowApplicationWebStart windowApplication = new WindowApplicationWebStart();
        windowApplication.setCommandLineValues(commandLineValues);
        startApplication(windowApplication);
    }
}
```

The arguments from the JNLP file are passed as normal command line arguments ‘args’ to the main program and used to start the application.
4.10 Identification and Start of an Assessment

The only way that ISAC is started so far, is via command line. After the client is started the user has to log in, before he or she can continue using the GUI of ISAC to browse through examples, problems, theories or methods in different categories and to start calculations from scratch or to solve them step by step or make ISAC solve the problem for them.

From US 6 ISAC needs to be automatically logged in by the credentials provided by § 4.9, it needs to load the environment according to some settings unique to the MASS and the assessment mode. But first we analyse the architecture we have right now.

4.10.1 Status-quo in the ISAC Maths Assistant

The following diagram describes the initialization of objects after the user correctly logged in.
ISAC is being started by command line by calling the WindowApplication. The login process requires the ObjectManager and the other server components to be running and then it proceeds as follows:

(1-2) Right after the start the LoginScreen is shown to ask the user for username and password. (3-4) After that the user data is entered, the “Login” button triggers a listener. (5) The ILoginListener calls the loginEventOccured method, which asks (6) if the type of the LoginEvent is of type LOGIN_TYPE_TRY_TO_LOGIN the system tries to login with the entered username and password otherwise if the user presses “Cancel” (19) the GUI will be closed and the program exits. (7-10) To Check username and password the UserManager (a singleton, part of the ISAC server) is called over RMI (running via the ObjectManager). (11-13) Via the login method of the UserManager a new account will be created. This is only the case if the
authentication is valid. This information is read from property files, which keep authentication and behavioural properties of the user interface separated. Then a new user object will be created for the new session:

```java
Accounts acs = new Accounts(userpath);
if (acs.authenticationValid(username, password)) {
    User user = new User(username, userpath);
    allLoggedInUsers.put(username, user);
    newSession = SessionManager.getInstance().addSession(user);
}
```

(14-18) Then the Session creates the BrowserDialogs (Problem-, Method-Theory- and ExampleDialog) and the WSDialogManager (see Figure 4.10.1a). (21) The BrowserDialog has to know its browser, but the browser is created later (within the WindowApplication). Therefore the browser is just registered to the dialog for now. This ends the initialisation and registration of the ISAC client to the ISAC server. Finally on the ISAC client the WindowApplication will be initialized and made visible to the user and the LoginScreen will be closed.

### 4.10.2 Detailed Design for the Instantiation

The former process in Figure 4.10.1a needs to be modified to automatically log in the user:
In (1) the ISAC client (the `WindowApplication`) now takes additional arguments: a username, password, assessment mode, example id. These arguments are transferred via RMI to the ISAC server and start the login process. (2-15) The username and password automatically log the user in. If the authentication is correct then the 'Login' dialog (`LoginScreen`) is not shown, otherwise it is made visible and an error message is displayed:

```java
currentSession = userManager.login(username, password, learningMode);
if (currentSession == null) {
    loginScreen.setVisible(true);
    loginScreen.showErrorMsgWrongUsernameOrPassword(username, password);
} else {
    currentSession.registerBrowserFrame((IToGUI) this);
}
```

However, additionally the assessment mode is also stored in the `Session` (16) which leads to different behaviour in the frontend (eg. a user may not browse the examples or autosolve problems, a.s.o, see US 8). (18) registers the browser to the `BrowserDialog` and in (19) the example id is transferred.
to the ISAC server and leads to opening a new Worksheet with the example that is to be solved by the user. The implementation of this functionality is actually part of § 4.11 (US 7) but already shown here in this diagram for completeness sake.

At this stage a new ISAC session will be opened for each HTTP request. This will change in the next chapter.

Another new concept was introduced with Domains. Before, ISAC could configure an environment for each user that is logged in. But for the access of many students through a MASS only one configuration for all students is needed. So it was decided that a MASS uses the username to authenticate to ISAC as a duple domain/studentId (or massId/studentId). ISAC only uses the domain part (and the password) for authentication of each MASS. The studentId is only needed to track the progress of the student and to create a unique session. In the UserManager the login method uses this username for authentication:

```java
public boolean isAuthenticationValid(String username, String password) throws RemoteException {
    User user = new User(username, ObjectManagerPaths.SESSION_MANAGER_USER_PATH);
    return authenticationValid(user, password);
}

private boolean authenticationValid(User user, String password) {
    if (user.hasDomain()) {
        Domains domains = new Domains(user.getUserpath());
        return domains.authenticationValid(user.getDomain(), password);
    } else {
        Accounts accounts = new Accounts(user.getUserpath());
        return accounts.authenticationValid(user.getUsername(), password);
    }
}
```

The hasDomain() method checks if the username consists of a duple like shown above and decides either to use the current authentication (no duple) where the username and password are being looked up in the accounts.txt or the domains.txt. In both cases the full username is stored in the session and the environment is changed by reading some definition files.

It must be noted that all changes were made in a way that the previous
functionality, like to start ISAC per command line, is still available.

4.11 *Execution of Assessment Items*

For this user story (US 7) it was desirable that when the ISAC client has already been opened as shown in § 4.10 and the user clicks another example that the new example should also open within the current ISAC client session. It should not open in a completely new session (new ISAC window).

**4.11.1 Status-quo in the Maths Assistant**

The following sequence diagram shows how a new Worksheet is being opened when a user clicked on a hyperlink in the ExampleDialog, within the ISAC client:

![Sequence diagram](image)

Figure 4.11.1a Sequence diagram for “Open an example”

After the user clicked on the hyperlink (1-2) within the ISAC client’s own example browser (BrowserFrameSwing) the contents of the clicked hyperlink is sent to the ExampleDialog via RMI through the notifyUserAction() to the ISAC server. Then (3) the link is interpreted in the BrowserDialog. Then in (4) an instance of the ExampleDialog is retrieved from the session. With the returned ExampleDialog (5-6) openWorksheetFromExample is called. This method prepares the example for the Worksheet. Now in (7-10) the
WSDialogManager is fetched to open a new WSDialog. Then the WSDialog sends (11) a doUIAction via RMI back to the ISAC client to open the requested example in a new worksheet. Finally (12-13) the WindowApplication generates a Worksheet with the example and shows it to the user.

4.11.2 Detailed Design for the Instantiation

The status-quo approach of opening a worksheet could not be reused since the user does not click on a new example within the ISAC client, but from a webpage outside of ISAC, within the MASS.

The main challenge here is how to remotely cause the example to open in the ISAC client just by clicking same URL again, since there is no way to directly control the ISAC client from the browser on the client computer. Two paths were possible:

a. The click on the URL would reach the application server and then download and open another ISAC client (like in Figure 4.10.2a), but would not show it to the user yet. Instead it would first try to find out if another client is already open on the users computer and then continue accordingly.

b. The click on the URL would only reach so far as the application server. From there the ISAC server will be contacted via RMI and asked if a session already exists, meaning if a ISAC Java frontend client for the current user has already been opened on the clients computer. If that is the case, a new worksheet will be opened in the existing ISAC client.

It was decided to use the latter approach:
When the user clicks on a link at the MASS for a new example, the link reaches the application server. There the URL is dispatched to the WebAccessAction, which got extended to do two steps now:

```java
try {
    objMan = (IObjectManager) Naming.lookup("RMI_OBJECT_MANAGER");
    webAccess = objMan.webAccess(webAccessForm.getUsername(),
                                webAccessForm.getPassword(),
                                webAccessForm.getMode(),
                                webAccessForm.getExample());
} catch (...) {
    // exception handling
}
...
if (webAccess.hasSessionToOpenExample()) {
    return mapping.findForward("SUCCESS");
} else {
    return mapping.findForward("JNLP_DOWNLOAD");
}
```

Now instead of downloading the JNLP right away it first creates an RMI connection to the ISAC server (1) (ObjectManager). Then, on the ISAC server side, the SessionManager is queried if a Session for this user already exists (2-3):

```java
public WebAccessResponse webAccess(String username, String password,
                         String mode, String example)
    throws AuthenticationException, RemoteException {
```
WebAccessResponse webAccessResponse = new WebAccessResponse();
if (user_manager_.isAuthenticationValid(username, password)) {
    Session session = dialog_.getFirstSession(username);
    if (session == null) {
        webAccessResponse.setHasSessionToOpenExample(false);
    } else {
        openWorksheet(session, example);
        webAccessResponse.setHasSessionToOpenExample(true);
    }
} else {
    throw new AuthenticationException("User '" + username + 
' or password '" + password + '" incorrect.'");
}
return webAccessResponse;

If no Session exists, a new one is created (see § 4.10.2 for details). If a Session exists then the ExampleDialog is called to load the example from the knowledge store (KESTore) (5-7). Then the WSDialogManager is called to open a new WorksheetDialog (8-11) for the already open Session and finally the ISAC client is contacted via RMI (12) again to display the new example in the Worksheet (13-14).

Back at the application server the returned WebAccessResponse now contains information if a session already existed. If it did, then the web page of the user is being redirected to the ‘SUCCESS’ page. If no session exists, then as before (see § 4.9) the browser is forwarded to the ‘JNLP_DOWNLOAD’ page (configured in the struts-config), which is served by the /WEB-INF/jsp/jnlpDownload.jsp file. The JNLP file is being downloaded and opened on the client machine and leads to opening the ISAC client.

4.12 Enforce the Assessment mode

It has to be made sure, that a user does not open a ‘normal’ learning mode session and an ‘assessment’ learning mode session at the same time, otherwise the user is able to cheat at the assessment. For this the functionality of the webAccess method of the ObjectManager above (§ 4.11.2) was extended to the following

    public WebAccessResponse webAccess(String username, String password,
    String mode, String example)
    throws AuthenticationException, RemoteException {
    WebAccessResponse webAccessResponse = new WebAccessResponse();
if (user_manager_.isAuthenticationValid(username, password)) {
    Session session = dialog_.getFirstSession(username);
    if (session == null) {
        webAccessReponse.setHasSessionToOpenExample(false);
    } else {
        if (session.getLearningMode().toString().
                   equalsIgnoreCase(mode)) {
            if (session.getLearningMode() == LearningMode.ASSESSMENT){
                openWorksheet(session, example);
                webAccessReponse.setHasSessionToOpenExample(true);
            } else {
                closeAllOpenSessions(username);
                webAccessReponse.setHasSessionToOpenExample(false);
            }
        } else { // mode != ASSESSMENT
            if (session.getLearningMode() == LearningMode.ASSESSMENT){
                closeAllOpenSessions(username);
                webAccessReponse.setHasSessionToOpenExample(false);
            } else { //session != ASSESSMENT
                openWorksheet(session, example);
                webAccessReponse.setHasSessionToOpenExample(true);
            }
        }
    }
    else {
        throw new AuthenticationException("User '" + username + 
        "' or password '" + password + "' incorrect.");
    }
    return webAccessReponse;
}

If a session exists in normal mode and a new assessment session is being created, all other sessions will be closed first (closeAllOpenSessions()). Part of the assessment mode is also to hide certain parts of the UI like the ‘next’ button, which tells ISAC to solve or simplify the current equation further, or the ‘auto’ button which solves the complete equation with all the steps in between. The implementation for this part is only a prototype and has only been made to showcase the scenario. In the WorksheetDialog (see Figure 4.11.2a) only one method needed to be changed to be aware of the learning mode:

private void addNextAndAutoButtonForWorksheet() throws RemoteException {
    if (session_.getLearningMode() != LearningMode.ASSESSMENT) {
        // Button next
        ui_control_listener_.addUIElement(new UIAction( 
            EUIElement.UI_SOLVE_CALCULATE_1, 
            EUIContext.UI_CONTEXT_CALCULATION, user_language_));
        // Button auto
        ui_control_listener_.addUIElement(new UIAction( 
            EUIElement.UI_SOLVE_CALCULATE_ALL,
This was the only place where the buttons were created in the code. However at the time of the implementation of this feature it had not been decided if the location for this change in the source code was the right one, from an architectural point of view. Also other functionality could be added, like removing the Browser buttons (Theorem, Example...), but the decision for this was deferred.

4.13 Summative Assessment

In the current stage ISAC is able to track every single step of what the user is doing within ISAC. ISAC itself does not exactly directly support summative assessment itself right now, however there are “heaps of different summative assessment” [Ber11] approaches. And through the information provided by ISAC the MASS is able to use this data for its summative assessment functionality. This functionality is called for in use case UC 3 (§ 2.3.1).

4.13.1 Status-quo in the Maths Assistant

When a user solves a problem in (applied) mathematics within ISAC, he usually needs several steps to proceed from the problem statement to the solution. Every step in this process is being tracked. The UserLogger is notified by each step and writes it into the userlogger table of the connected database:

<table>
<thead>
<tr>
<th>userlogger_id</th>
<th>username</th>
<th>session_id</th>
<th>dialog_type</th>
<th>worksheet_id</th>
<th>time</th>
<th>position</th>
<th>formula_from</th>
<th>step</th>
<th>step_arg_1</th>
<th>step_arg_2</th>
<th>formula_to</th>
<th>error</th>
<th>error_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>Mono</td>
<td></td>
<td>2012-04-11</td>
<td>null</td>
<td></td>
<td>UD START SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>0</td>
<td>Mono</td>
<td></td>
<td>2012-04-11</td>
<td>null</td>
<td></td>
<td>UL.BROWSER_CHANGE_CONTEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>0</td>
<td>Workbook</td>
<td></td>
<td>2012-04-11</td>
<td>null</td>
<td></td>
<td>UL.ORIG WORKSHEET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.13.1a The UserLogger table in the database

Each entry gives exact information of who (via session_id and username) solved at a certain point (userlogger_id, time) which equation (via worksheet_id, formula_from, formula_to).
4.13.2 Detailed Design for the Instantiation

The application server that has already been used in § 4.10 and § 4.11 can easily be extended to deal with REST calls\(^{68}\). Nowadays, creating a webservice is fairly simply accomplished. Many frameworks that deal with the complicated details have already been created. Here Oracle’s jax-ws\(^{69}\) technology and the todays IDE’s ability to automatically create a jax-ws webservice from a database schema is used for rapid development. Then the webservice has been fine-tuned to the particular needs.

The following diagram shows some of the objects that are used to receive a REST call, requesting all progress information about user ‘x’, and return the data as JSON/XML:

![Diagram of the MASS retrieving progress information via REST call.](image)

Figure 4.3.2a: The MASS retrieves progress information via REST call.

The MASS calls (1) the application server by making a synchronous REST call over HTTP to the application server. The application servers container (jax-ws service) receives and dispatches the request according to the URL path (eg. /services/progress/user) and the applications configuration to the UserLoggerRESTFacade (2) and calls the defined method (see below). Then, from the EntityManager of the containers JPA\(^{70}\), the named query\(^{71}\) is created (4-11) which looks up from the UserLogger’s annotation which query should be made to the database. Into this query the username is set.
(12-13), which has been resolved from the URL in (2). In step 14-17 the final query is been transformed from JPA notation to SQL notation, the database is queried and the result set is transformed into a list of UserLogger objects. Finally (18) the list of objects is being transformed into a JSON/XML string, which is returned to the MASS. The final XML looks something like this:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userloggers>
  <userlogger>
    <dialogType>NONE</dialogType>
    <error>false</error>
    <position>null</position>
    <sessionId>0</sessionId>
    <step>LO_START_SESSION</step>
    <time>2012-04-23T23:12:12+02:00</time>
    <userloggerId>1</userloggerId>
    <username>x</username>
  </userlogger>
  <userlogger>
    <dialogType>EXAMPLE</dialogType>
    <error>false</error>
    <formulaTo/>
    <position/>
    <sessionId>0</sessionId>
    <step>UI_BROWSER_CHANGE_CONTEXT</step>
    <stepArg1>miniscript_with_mini_subpbl</stepArg1>
    <stepArg2>Examples\IsacCore\Tests</stepArg2>
    <time>2012-04-23T23:22+02:00</time>
    <userloggerId>2</userloggerId>
    <username>x</username>
    <worksheetId/>
  </userlogger>
  ...
</userloggers>
```

The MASS is then responsible to analyse the data and use it for its summative assessment.

To go a little more into detail consider this example URL

```
http://myhost.com:8080/isac-web/services/progress/user/x
```

How the first part of the URL is mapped to the Glassfish application server has already been described in §4.9. Now, the configuration for the /services URL path has been added to the web.xml. For this path a servlet is configured, which looks up for the REST service handler in the
org.isac.service java package. Currently only one handler exist there, called **UserloggerRESTFacade**:

```java
@Stateless
@Path("progress")
public class UserloggerRESTFacade extends AbstractFacade<Userlogger> {

    @PersistenceContext(unitName = "isac-webPU")
    private EntityManager em;
    public UserloggerRESTFacade() {
        super(Userlogger.class);
    }

    @GET
    @Path("id/{id}")
    @Produces({"application/xml", "application/json"})
    public Userlogger findById(@PathParam("id") Integer id) {
        return super.find(id);
    }

    @GET
    @Path("user/{username}")
    @Produces({"application/xml", "application/json"})
    public List<Userlogger> findByUser(@PathParam("username") String username) {
        return super.findByUserName(username);
    }

    @GET
    @Path("user/{domain}/{username}")
    @Produces({"application/xml", "application/json"})
    public List<Userlogger> findByDomainAndUser(
        @PathParam("domain") String domain,
        @PathParam("username") String username) {
        return super.findByUserName(domain + "/" + username);
    }
    ...}
```

Here it can be seen how simple a REST service can be created. It defines all parameters for this service. For example that this Facade provides operations that can be reached under the URL path `/progress`. There are currently only query operations supported, for `id` or `user`. The detailed queries are defined in the **UserLogger** class in form of named queries, which look like SQL statements, eg.

```sql
SELECT u FROM Userlogger u WHERE u.username = :username
```
It also defines where the entity manager can be found: ‘isac-webPU’. This one is associated with the ‘jdbc/userlogger’ datasource in the persistence.xml file. And this datasource (the connection to ISAC’s UserLogger table in the database) is configured in the Glassfish server’s glassfish-resources.xml file, thus keeping the definition and usage of the data source separate.

After the web application has been started via Netbeans, a web browser is being started, showing a web page with example links that can be used in a MASS for these two scenarios. This page is being generated from the /WEB-INF/jsp/index.jsp file.
5. Summary and Conclusions

This master thesis concerns a sophisticated combination of web technologies in order to provide an interface between an assessment-engine and a general class of tools for problem solving. Instantiation of the interface was demonstrated on a mathematics assistant prototyping a new generation of educational mathematics tools.

Summary of the thesis’ structure

§ 1 introduced the two software components to be combined, the web based modular assessment system MASS on the one side and the mathematics assistant ISAC based on Computer Theorem Proving on the other side. This chapter also stated the goals, which all are now reached by the end of the thesis. In § 2 the concepts, technologies and components for a variety of web interfaces between the MASS and ISAC were identified. This involved a review of the respective literature on the technologies at the present state-of-the-art and an evaluation of which components were to be used. § 3 reviewed these web technologies in order to achieve an architectural design combining MASS and ISAC. This posed challenges, since ISAC’s architecture and technology did not allow straightforward web-access. However, we found a feasible adaption of ISAC’s architecture to a general and elegant web-access. § 4 made the requirements explicit, provided a detailed design and a working implementation. Both are described in detail such that subsequent development easily can find the entry points.

Summary of the thesis’ practical work

A feasibility study [Zillinger 2013] preceded the practical design and implementation work; it had identified the major design questions. The most challenging one was how to start within a web based system on one web domain another application that could only be started per command line and runs as a desktop client on another domain. Another challenge was how to
pass authentication, student and exam specific information to the remote system during startup, such that the student is to be logged in immediately and he ultimately could continue with the exam right away. Other problems appeared during the conceptional and also in the architectural design stage, like who should be in charge of user management or how can cheating be prevented by starting the mathematics assistant a second time in ‘normal’ mode and thus being able to use the ‘auto solve’ mode for the exam. Of course there were also technological challenges. The strong legacy architecture first seemed to prevent a possibly simpler solutions, but in the end however this strong architecture encouraged a cleaner one.

The practical work within the master thesis reflects the agile nature of development required by a prototype like ISAC. In order to cope with the complexity of the system and the realisation of the findings and plans from the feasibility study, the practical work started with user stories, some derived from the use cases, some addressing the challenges to handle the variety of technologies used --- the latter was requested by the supervisor in order to support further development.

What still was vague in the feasibility study turned out useful during the practical work; the key points of the architecture finally realised are depicted in Figure 3.2a: a desktop client, three server instances, all together in one monolithic application that connects to a database and to an external theorem prover instance and also retrieves data from an external knowledge store.

**Conclusions about the advancements accomplished by the thesis**

In this master thesis the following accomplishments were made:

- A detailed analysis of the use cases, an analysis of the current architecture of ISAC and the MASS and a design for a web interface connecting ISAC and MASS.
- A comparison and explanation of state-of-the-art Web technologies
relevant for realising the architecture.

- An implementation of a Java Web Start solution for the existing ISAC desktop client that connects with its remote server. Access control is designed for flexibility: anonymous access (for instance from a math wiki), access for groups of learners (as administered by some LMS) as well as access for written exams.
- The access for written exams allows to prevent cheating by connecting to the server in non-assessment mode (which might lots of help for finding the solutions).
- Building an application server, that allows to dynamically change the Java Web Start configuration including authentication, and to change student and exam related information to be passed to the desktop client on startup
- The application server is extended with web service functionality that allows to download the progress information of the student.
- Some functional code stabilisations, like code refactorings during design and implementation, which reduced errors and increased stability in the start and running phase of ISAC.

**Conclusions about ISAC’s future development**

Already the feasibility study identified some interesting requirements as future work. The actual implementation work within this thesis was more demanding than expected; so the status as "future work" is confirmed for these points:

- ISAC records all high-level steps in problem solving, not only the final result. This master thesis only used the latter; obviously summative assessment could be more valid, if some other steps would be taken into account as well.
- The above mentioned steps, which ISAC records during problem solving, could be used for formative assessment and user guidance: ISAC’s dialog engine is already determined by an external rule engine, therefore using the database as a source for user history and
combining both external modules (the rule engine and the data base) for adaptive user guidance seems straightforward.

- The XML that is provided by the application server (in § 4.13.2) for the MASS should be specified and versioned. Consequently changes of the XML data due to further development could be recognized by the MASS and adapted accordingly.
- The new ‘assessment’ mode with its restrictions needs to be reflected in the user interface, for instance the ‘Auto’ and ‘Next’ buttons, which help and guide the student in the process of solving equations, should disappear during ‘assessment’ mode. Also The Browser buttons (Example, Theorem,...) could be removed.
- Automatic generation of a signed Java Web Start application would make deployment easier. Presently signing the application has to be done manually or the JVM has to be configured separately in order that an RMI connection between ISAC client and server can be established.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>ARPANET</td>
<td>Advanced Research Projects Agency Network</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>Facade</td>
<td>The facade pattern</td>
</tr>
<tr>
<td>Glassfish</td>
<td>An open-source application server</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
</tr>
<tr>
<td>IICM</td>
<td>Institut für Informationssysteme und Computer Medien</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet message access protocol</td>
</tr>
<tr>
<td>Isabelle</td>
<td>A generic proof assistant</td>
</tr>
<tr>
<td>ISAC</td>
<td>An experimental mathematics assistant</td>
</tr>
<tr>
<td>IST</td>
<td>Institute for Software Technology</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java Platform, Enterprise Edition</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML Web Services</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistance API</td>
</tr>
<tr>
<td>JSF</td>
<td>Java Server Faces</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JNLP</td>
<td>Java Network Launching Protocol</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>JWS</td>
<td>Java Web Start</td>
</tr>
<tr>
<td>MASS</td>
<td>Modular Assessment System for Modern Learning Settings</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Control</td>
</tr>
<tr>
<td>MySQL</td>
<td>An open-source relational database management system</td>
</tr>
<tr>
<td>Netbeans</td>
<td>An open-source integrated development environment</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor</td>
</tr>
<tr>
<td>POP3</td>
<td>Post Office Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet applications</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation protocol</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SVN</td>
<td>Apache Subversion</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>TP</td>
<td>(Computer) Theorem Proving</td>
</tr>
<tr>
<td>TU Graz</td>
<td>Graz University of Technology</td>
</tr>
<tr>
<td>UC</td>
<td>Use case</td>
</tr>
<tr>
<td>UI</td>
<td>User interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>US</td>
<td>User Story</td>
</tr>
<tr>
<td>Virtualbox</td>
<td>Oracle VM VirtualBox</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XHTML</td>
<td>Extensible HyperText Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
References


