ABET Student Certification Program Website Evaluations

Project Documentation for HCI Project II (HCI 551)

submitted in partial fulfillment of requirements of the degree

Master of Arts in **Human-Computer Interaction**

December 13th 2008

Submitted by[*] Bastian Tenbergen[‡]
Wayne Weibel[‡]

Supervisor Prof. Dr. Douglas Lea[†]

† dl @ cs.oswego.edu

‡ { tenberge | weibel } @ oswego.edu

* in alphabetical order

Human-Computer Interaction MA Program
Department of Psychology
State University of New York, College at Oswego
Oswego, NY, 13126

Abstract

ABET is a certification program for Software Engineering students. Since the State University of New York at Oswego has recently started a new Software Engineering major, a website needs to be developed that accompanies the program certification. This is the second project for the submitting students in fulfillment of the degree requirements for the degree they seek to obtain and as such, a project with focus on product evaluation. Hence, this project will develop and evaluate a set of high-fidelity prototypes that can be used as immediate guidelines for the implementation of such a website.

Table of Content

SUNY Oswego Software Engineering Program ABET Accreditation Website Evaluations

1. Part I – Project Description and Scope

This section gives a general overview over the project. The introduction will provide a detailed description of the project motivation and the certification program, for which interface prototypes needed to be created. Furthermore, the procedure and timeline of this project is outlined and the distribution of responsibilities will be discussed.

1.1 Introduction

SUNY Oswego is incorporating a Software Engineering major and a directive of the program is to receive and maintain ABET accreditation. ABET provides world leadership in assuring quality and in stimulating innovation in applied science, computing, engineering, and technology education. In the United States, accreditation is a non-governmental, peer-review process that assures the quality of the postsecondary education students receive. Educational institutions or programs volunteer to undergo this review periodically to determine if certain criteria are being met. This accreditation is based upon the curriculum and objectives of the program, not solely the performance and production of the enrolled students. The effectiveness of the program will be determined by the quality of the objectives and lessons designed to meet them. These in turn are measured by the performance of the students; the averages of quizzes, assignments, and the success of projects.

ABET accreditation is not simply a raw mathematical score though; the quality of learning that is incurred is weighed more heavily than just the final grade. With this in mind a way to track the performance of the students as a function of the program objectives is required. A site for the faculty to record project descriptions, lessons, as well as, student scores is intended, with the possibility of selecting samples of student work.

This project will entail gathering the necessary preliminary data to formulate a strategy for addressing the needs of the site. These will then be translated into, at minimum, a prototype of the site. User testing and evaluation will be run against this prototype and any subsequent versions as a result of the testing and evaluation cycle. Ideally, a high fidelity prototype will be developed, which will allow for more in depth testing too take place, including the use of the eye tracking device and software. This document describes how we will implement this project.

1.2 Procedure

Due to the fact that the ABET website has not yet been implemented a functional prototype needs to be developed from which the actual site design can be implemented. This project therefore is structured into two phases; the design phase and the evaluation phase, which will be run concurrently in a R.A.D. development cycle.

The design phase will entail the gathering of necessary requirements by meeting with project stakeholders and potential users of the site. As stakeholders, we identified the director of the Software Engineering program, Randy Odendahl, as well as Douglas Lea, the chairperson of the Computer Science Department. Primary users of the site will be the faculty associated with the program: it is there responsibility to develop comprehensive lessons, projects and grading rubrics that effectively track student performance. These are the basis for ABET accreditation.

Secondary users will be the enrolled students, who will benefit from receiving a degree from an ABET accredited science program.

To further investigate the requirements for the site design, classical user-centered requirements assessment strategies will be employed. These are, for instance, card-sorting assessments, contextual inquiries and expert interviews. The design phase will result, at minimum, in a set of high fidelity prototypes which can then be used in the evaluation phase. The high fidelity prototypes may be functional HTML sites or a non-functional mark-up of a possible site designs using design tools.

The evaluation phase aims at identifying problems in the site design. Since the site serves as a platform to host resources for students as well as students' projects and degree progress, special emphasis during the site design must be placed on disambiguation and clarity of the site's content. Furthermore, the site must be easy to navigate. These properties will be evaluated using classic user-testing methodology; by making use of eye tracking and video recording technology user performance can be analyzed to identify potential pitfalls of the site design. The evaluation of the site will serve as a guideline in the production of the final revision of the SUNY Oswego Software Engineer ABET website.

It shall be clear, that this project is an evaluation project. The main focus will be the evaluation of different design choices. These can then be used to make informed decisions for a final site design. Designing the actual, deployable website is beyond the scope of this project.

A preliminary and optimistic schedule follows.

1.3 Timeline

The project will adhere to the following, preliminary schedule.

	Begin design phase.
08 / 27 / 2008	Preliminary meeting and outline of site needs and requirements. Begin development of survey questionnaire and card sorting items.
09 / 01 / 2008	Administer surveys and card sorts.
09 / 05 / 2008	Gather design guidelines and rubric choices from surveys, begin evaluating data.
09 / 12 / 2008	Finish analyzing data. Begin development of project documentation. Design initial site layout and concepts.
09 / 19 / 2008	First inception of Low Fidelity prototype. Applied to Human Studies.

	Begin evaluation phase.
09 / 19 / 2008	Last date to receive approval from IRB/HSC for testing.
09 / 26 / 2008 to 10 / 03 / 2008	Test and Evaluate first iteration of low-fidelity prototypes.
10 / 10 / 2008	Finish re-designing of prototypes on basis of preliminary findings.
10 / 17 / 2008 to 10 / 24 / 2008	Test and Evaluate second iteration of low-fidelity prototypes.

10 / 31 / 2008	Finish re-designs of prototypes. Begin high-fidelity prototypes.
11 / 07 / 2008	Finish development of high-fidelity prototypes. Reapplied to Human Studies.
11 / 14 / 2008	Testing with Eye Tracking equipment of high-fidelity prototypes.
11 / 28 / 2008	Thanksgiving Recess
12 / 05 / 2008	Complete evaluation of data from high-fidelity prototypes testing.
12 / 19 / 2008	Project Deadline – Final Report Due.

1.4 Responsibilities

This project was jointly prepared by Wayne Weibel and Bastian Tenbergen. The responsibilities of the submitting students will be evenly shared amongst them. At this time, there are no precise plans regarding the distribution of the work load; future project documentation will identify special tasks as they arise and which submitting student assumed responsibility (in full or in part) of completing the task. An effort will be made to evenly distribute tasks among all submitting students with regard to work load and level of complexity.

2. Part II – Design Rationale and Prototype Implementation

This section of the Project Documentation describes the design phase of the project. A list of requirements was compiled from the meetings with the project stakeholders and implementation details and design choices have hence been made. These are explained herein and served as the fundamental implementation principles according to which this project has been completed. Since these principles and design choices largely influences the evaluation part of this project (which also is the focus of this project), it was important to place special emphasize on the design phase so that accurate evaluation of competing prototype alternatives can be made and their individual pros and cons can be assessed.

2.1. System Requirements

The system requirements are largely dependent on the certification requirements that ABET has for candidate programs. In order for a candidate program to become ABET certified it must satisfy a number of eleven identified guidelines. The proposed goals of the guidelines rarely changes, but may be adjusted in the future nonetheless. Candidate degree programs must consist of a set of objectives, which fulfill one or more of these ABET guidelines. In turn, for a student to graduate from a ABET certified program, it is necessary for the student to adhere to and complete the degree program objectives that fulfill all ABET guidelines.

Since the degree program objectives are currently not well-formed and still under review it is necessary for the proposed certification review website (which is the subject of this project) to allow faculty to adjust the program objectives, as well as, the ABET guidelines they fulfill. Also, the system must make available the ability to maintain records of which students have fulfilled which degree objectives. The fulfillment of the degree objective can either occur by a student completing a course (that fulfills the degree objective) with a passing grade (corresponding to a letter grading rubric), by showing proof that knowledge in this degree objective is sufficient (pass/fail grading rubric) or by some artifact of completed work, such as a paper, independent study, project or the like (corresponding to an artifact grading rubric).

It hence becomes clear that the system must be designed as a raw framework, in which ABET guidelines, degree program objectives, grading rubrics, and students can be added and removed, altered or changed dynamically. The interaction with the website must therefore be designed accordingly.

From these requirements, the question arises, why the system cannot be designed as a desktop application. The main goal of this project is to develop a central point where degree program faculty and ABET certification authorities can monitor the certification requirements of the degree program and students can monitor their personal degree progress, therefore, a downloadable and deployable desktop application poses problems that must first be addressed.

First, communication with a database must conform to certain security standards – a necessity that can be addressed in a website paradigm, as the web server will be the only trusted node to access a database. Authentication can therefore be performed on a per-user basis rather than on a per-machine and per-user basis, as it would be in a desktop application. Furthermore, ABET certification officers will need a quick and easy way to access the data and review the certification process; deploying a desktop application for that purpose is posing integration issues with a potential significantly differing platform and would hinder an easy interaction.

2.2. User Groups

Initial meetings with the project stakeholders have identified that there will be three different, but equally significant user groups. These are, firstly, SUNY Oswego faculty (the primary user) who will maintain the program overview regarding the ABET certification requirements, degree program objectives and the students fulfilling these objectives. These users must be able to edit the entire content of the site. The secondary user group are ABET certification officers who review the degree candidate program at SUNY Oswego and maintain an overview of which program objectives fulfill what ABET guideline and if ABET certification can be awarded. ABET Reviewers do not need to edit content – it is sufficient for them to view the material. Lastly, the tertiary user group is the enrolled students. They will use the site to review their degree progress and also need only viewing abilities with no editing rights.

The following are representative of typical personas for each user group:

2.2.1 Persona: SUNY Oswego Faculty of Software Engineering Program (Primary User)

They are responsible for developing the rubrics for scoring the success of the students in satisfying the requirements of the program for ABET accreditation. The program objectives are mapped to the ABET requirements in a many-to-many relationship, so there may be more or less than eleven requirements (which are deliberately kept vague by ABET). The faculty will be updating existing rubrics and submitting new approved rubrics. The site will also be used to track the actual performance of the students, so the faculty will be submitting the scoring for the students (based on the rubric of the course) as well.

Faculty member Dr John Raffers returns to his office after teaching his second class of the day. He has a short break before needing to attend a meeting with a group of students. Since it is nearing the end of the semester, he intends to use this time to update the ABET site with the students scores in their courses. He logs on to the site and is presented with a list of courses. After selecting the relevant course, a brief description of the rubric is displayed along with a list of the students currently attending the course. In the 212 course the professor quickly checks off whether a particular student has satisfied the requirement. In the 380 course Dr Raffers uploads the source documents of the students work. And in the 500 course the professor records the exam scores of each student. In addition, the professor views the rubric for the course they will be teaching during the summer session before logging out and heading to the meeting.

- Clearly designated sections
- Easy navigation between sections
- Minimal noise surrounding the information
- Easy upload of needed documents

The front page of the site displays general information with a clear log in port. Once logged in, the sections for rubrics and student scoring are clearly defined and the navigation to retrieve and submit information is minimal. The presentation of the student section will be based on the type of rubric being used. As well as the submittal of student scoring, there will be a reporting function in the form of a checklist showing the program requirements and the score for each satisfied requirement for each student.

“My time is very short, so being able to quickly accomplish what I need to do is very appealing.”

Dr John Raffers is 43 years old with over a decade of experience in teaching at the collegiate level. He spends over 9 hours each day on campus, 5 of which are spent in class, the rest in meetings with either students or faculty. Once home he spends more of their time reviewing the student projects and scoring their exams. Time is a valued commodity for this professor.

2.2.2. Persona: ABET Reviewer (Secondary User)

They are responsible for validating the accomplishment of a scientific program for receiving ABET accreditation. They must review the documentation describing the actions and rubrics being implemented by the faculty of the program, as well as, the performance of the students. The site will be utilized primarily as a reporting function; being used to view the rubrics and student work. And as such the program artifacts will be reviewed for accreditation a minimal number of times per year. They assess how well the program has satisfied the ABET requirements with their own objectives, course offerings and plans, and scoring rubrics.

Lorenza Hitchens is a ABET certified reviewer visiting the final of five universities in seven days. She is already looking forward to returning home for an extended break. She arrives at the main office building of the university and is met by the program chair and an accompanying faculty person. She is lead to and shown the facilities of the program, explained to the infrastructure, and told about some of the recent changes to the program. Now she needs to review the documentation of the program. The reviewer dreads this portion due to the amounts of paperwork involved, but in this case she is pleased to see that everything is represented digitally. She is directed to a website, which allows them to navigate between the course offerings and descriptions, view the rubrics being used, and even more conveniently are able to view the student performance in the courses, as either, satisfactory completion, numerical grades, or project artifacts. The implementation of this site greatly reduces the amount of time the reviewer needed to spend at the university.

- Clearly designated sections
- Easy navigation between sections
- Minimal noise surrounding the information
- Clearly formatted reporting pages

The front page of the site displays general information with a clear log in port. Once logged in, the sections for rubrics and student scoring are clearly defined and the navigation to retrieve and submit information is minimal. The presentation of the student section will be based on the type of rubric being used. As well as the submittal of student scoring, there will be a reporting function in the form of a checklist showing the program requirements and the score for each satisfied requirement for each student.

“The amount of documentation that needs to be reviewed is daunting, so having all the necessary information quickly and clearly accessible makes the process much simpler.”

Lorenza is 47 years old, an accomplished educator in Computer Science, living in Arizona. She spends the majority of her year traveling between their home, the university where she has tenure, and the universities that are up for ABET accreditation (renewal or initial). With their precious free time they use it to read or spend it outside.

2.2.3. Persona: Software Engineering Student (Tertiary User)

They have no direct interaction with the site, as far as, submitting grades or work or influencing the mechanisms used in scoring their work. Depending on the success of the program in meeting the ABET requirements, and on them in turn meeting the program objectives, they will receive ABET certification. The site will be utilized as a checklist to ensure that they have met all of the required objectives to receive this certification.

Marcus Eisenhardt is entering his senior year and preparing to register for his classes. The program is ABET accredited, but he is unsure whether he has satisfied the program requirements to receive their own certification. He goes to the SUNY Oswego ABET site, logs in, and is presented with a list of the objectives, with an indication of the ones they have satisfied, and are able to quickly see that there are two courses that he must register for in the fall semester since they are not offered in the spring. He prints off the list and logs out.

- Easy log in
- Clear and expandable objective sections
- Clear indication of objective completion
- Printing ability

The front page of the site shows general news pertaining to the program and clear log in port for the site. Once logged in there is no navigation necessary, the checklist for the objectives along with completion indication and submitted work (if applicable) is displayed. The options to Print or Log Out of the site are clearly represented.

Marcus Eisenhardt is 21 years old, an entering senior in the Software Engineering degree. He is a Long Island resident and heavily involved with CSA, the Computer Science Association. He has frequent internships in New York City during semester breaks. He is attempting to implement a natural language processor using neural networks as a free time project and like to interact with his faculty when he has academic questions.

2.3 User Needs Analysis

Analyzing the users' needs is essential in identifying the features that the system must be able to perform and that the interface must offer to the users. However, assessing these needs is a double-edged process – it is necessary to quantitatively measure and record what actions are executed by the users, as well as, how the user expects to accomplish those actions. On the other hand, as is often the case, there exist features that the users do not know they need or even want. This can not be quantitatively measured, but must be assessed through a healthy portion of intuition, experience, and observation.

However, by filling the front end with options and features that are only rarely used, or not needed at all, will lead to a crowded, convoluted and confusing interface, as a result, actually hindering the user experience and the effectiveness of the interface. It is of major importance to keep the interface simple and stream-lined. We hence conducted a series of semi-structured interviews with the project stakeholders to assess the basic needs and requirements of the users. Also, the personas helped in identifying potential pitfalls in user interaction by showing what minimal features must not be absent from the prototypes.

From the semi-structured interviews and the user identification that lead to the personas in the earlier section, we concluded that the priority of the interface features shall be on content modification (which includes adding, deleting, or changing of rubrics, objectives and guidelines). The prototypes developed for this project therefore will include the ability to add ABET guidelines, degree program objectives (along with their grading rubrics) and students. Also, ease of modification of the mapping between the guidelines and objectives will be a reasonable requirement for the interface, since these mappings may change frequently. This is due to the nature of the prototype domain being widely flexible and specific details about mappings from ABET guidelines to degree program objectives changing quite frequently in the future.

Additionally, information presentation is of prime importance for all user groups. Especially for the primary and secondary user group, it is important for the prototypes to allow a quick, but thorough overview of the content. We assume that the success of different prototype alternatives will be mostly dependent on their ability to present the underlying information efficiently and effectively.

2.4 Interface Design Alternatives

Given the user descriptions from the personas and the user needs analysis, it becomes obvious, that a high level of flexibility in the presentation of the content must be possible. Since two user groups (secondary and tertiary) will only be viewing the content, while the primary user group will mainly edit the material, a web site that allows for both secure presentation and dynamic modification of the content must be developed. We hence propose three different prototype implementations: an implementation as an Angel group, a semi-static Java Applet, and a GWT implementation of the front end.

The online course management system Angel allows the support for groups in which members of faculty and students can interact and submitted material. Since Angel is powered by a back end database, and takes responsibility to maintain user authentication and editing rights, it is a good platform for the implementation of the ABET certification website. It is only necessary to enter the content manually – one does not need to be concerned with the creation and maintenance of a back end database. However, Angel is somewhat restrictive in the type of content that can be added and users logging in to review the degree program certification must have a user account with SUNY Oswego. This will make it necessary for ABET Reviewers to obtain a user account at SUNY Oswego prior to review. We will evaluate this prototype to gather an understanding of how the limitations and advantages of Angel can help implement the ABET certification review website.

A Java applet as a second design alternative was chosen, as it can be conceptualized as a hybrid between a deployable desktop application and a database powered web site. Also, maintenance and modification of the Applet's user interface can happen at a centralized location. Further, applets allow for a form-like design of input widgets and can hence present the content in a dynamic manner by keeping the overall structure simple; implementation of the interface is rather simple and quick. It can therefore be easily changed to adapt to any arising user needs. There are, however, certain disadvantages. Java applets require additional software (outside of a current browser) to be installed on the client computer. Although Java is widely distributed and should be available on all modern computers, it might cause an extra layer of potential problems to arise; should a Java version be installed on some client machine that is incompatible with the applet. Also, Java applets are widely assumed to be unaesthetic. Since no modern interface design capabilities, such as 3D-widgets, anti-aliased fonts, transparency, etc, can be used (or require additional software to be installed on the client machine), the interface may appear rather somber and unappealing. We will evaluate this option to understand how an appealing design and the ability of applets to display database-driven dynamic content can provide the functionality necessary for an ABET degree program certification.

The last interface prototype alternative we chose is a GWT (Google Web Toolkit) implementation of a dynamic website. GWT allows for the use of conventional Java code to implement a website that is entirely dynamically generated upon a client request. It works by converting the Java code into a set of JavaScript functions that can be used to enhance the HTML content of an actual website. Since it is AJAX powered and has massive database support, it is often used for Web 2.0 in-browser applications throughout the web. It was therefore necessary as a choice to implement the ABET certification website. Due to the dynamic nature of GWT, it is able to create appealing websites that can give full dynamic access to the content. However, since this requires a server-side rendering of the website, it is necessary to have a web server that is able to deliver such content. We will evaluate this prototype choice to identify the benefits over other prototype choices and potential pitfalls in the user interaction.

2.5 Website Database Back End

The Java applet and GWT prototypes both require the presence of a back end database to establish basic functionality. Basic functionality is necessary for the evaluation phase – without at least fundamental functionality, the font end will merely be a static version of what can be done using the web site. Yet, due to the massively dynamic nature of the site content and the mission statement, dynamic interaction rather than a static one is absolutely critical and necessary during the evaluation of prototypes.

The database is by no means the focus of this project. Hence, in this section, only a brief description is given. For more details, please refer to Appendix A, which discusses details about the relation schemes, possesses an entity-relationship diagram and analyzes the normalization and normal form of the database.

For the database server MySQL has been chosen, this is due to the ubiquitous availability and the simplicity of setting up databases using that software. The web site front end will use the database to manage the content of guidelines, objective, performance criteria and performance items. Hence, the database must support a high level of dynamic content management and the ability to concurrently add, delete, or modify content, along with the ability to manage one-to-one, one-to-many, and many-to-many relationships between entities in the database. We hence chose a database design in Boyce-Codd Normal Form, as BCNF reduced the amount of redundancy of the stored data and also allows for quickly modifying the entity relationships with a minimum of computational overhead.

3. Part III – Prototype Evaluations

The following section will outline the evaluation plan of the project. Since this prototype is an evaluation project, this section therefore constitutes an essential part of the documentation. An explanation of participants and employed methodology will be given before the results of the evaluation of all project components will be presented. The final part of this section will give a design recommendation for this project, based on the previously discussed results.

3.1 Evaluation Plan

In contrast to the initially proposed timeline of the project, the evaluation of this project took place in two phases. The first phase entailed a thorough investigation of the principles of information interaction that have been proposed by the project stakeholders. This was very essential, as it immediately influences the implementation of the interfaces; this project requires a complex interaction of the mappings between different information items, such as ABET Performance Indicators, Software Engineering Program Objectives, Courses, and Students. Performance Indicators are in a many-to-many relationship with Program Objectives, meaning that one or more Program Objectives are fulfilling one or more Performance Indicators. Furthermore, Program Objectives are in a similar relationship with Courses, with the difference that Program Objectives are by proxy in a many-to-many relationship with Students. Every Student is enrolled in at least one Course, thereby being in a relationship with Program Objectives that are fulfilled by one or more Courses. The thereby instantiated relationship graph is hence of a complex, circular, bi-directionally feeding nature. Every node on that graph (which is an information item such as a Performance Indicator or a Course) stands in a many-to-many relationship with other nodes. This interaction of the information nodes was modeled by the project stakeholders, namely by the faculty of the Computer Science department of SUNY Oswego. The aim was to propose a visualization of the interaction such that it is easy to understand and modify without extensive knowledge of graph theory.

The second phase of evaluation was concerned with the evaluation of the interface prototypes. Since the prototypes seek to visualize the information in the most efficient manner, the measurement of efficiency is immediately dependent on the way the information is structured. The first phase has established the interaction concept, so that the second phase concentrates on how this interaction is implemented. This project proposes two different prototypes that are technologically different, in such that they employ different Java-based technologies – an Applet and a GWT Web Application. Both technologies have different possibilities and limitations to implement the certification tracking interface that is called for in this project and hence are conceptually different. Phase two therefore aims at usability and understandability of the interface alternatives; as the usability of the information interaction scheme has already been evaluated in the first phase.

It can be seen from the proposed timeline, that the evaluation phase was somewhat truncated than initially conceived. As this project was also very implementation heavy, the evaluation had to be adjusted to entail a more thorough analysis of interface features. Therefore, the submitting authors decided to focus the evaluation on the interaction scheme of the information items and the prototypes' ability to address this scheme and present relational data. Furthermore, due to research scheduling conflicts, the eye tracking device was not available for student use in such an extensive manner that late in the project. Therefore, it was decided to deepen the analysis of interface components on the basis of structured interviews and contextual inquiries, rather than eye tracking studies since the results were expected to be identical.

3.1 Participants

In the first phase, all participants were naïve to the study, but were briefly introduced into the ABET certification process and the implications for IT majors on higher education campuses. Six students have been recruited while lounging in an academic building on the SUNY Oswego campus and have been invited to participate. After their age and sex have been established, the students completed a consent form and were then lead for participation. An effort was made to counter-balance against age, sex and academic level (freshmen, sophomore, junior, senior, or graduate student).

In phase two, four professors of the SUNY Oswego Computer Science program were selected to evaluate the interface prototype alternatives. The choice of participants was extremely beneficial, as all four participants were identified as stakeholders of the project – these professors will be those using the interfaces to maintain and review the requirements and process for the Software Engineering major to receive ABET certification and therefore will be the ones interacting with the interfaces proposed by this project. The choice of participants was further fortunate in that two of the members are experts in usability and Human Computer Interaction and, more importantly, the Google Web Toolkit (and are therefore aware of benefits and limitations of said technology). Another of the participants is an expert on Database Management Systems and database front-end interfaces, and the last participant is an expert on the ABET Certification Specifications. All participants are male and have considerable knowledge in software implementation and engineering as well as administrative responsibilities and all are considered authorities in their respective fields as both scientists and pedagogues.

3.2 Methodology

Evaluation of phase one was conducted on a Hewlett-Packard Compaq TC4200 Tablet PC laptop computer. The tablet functionality was not used. The online course platform Angel was used to model the static HTML pages, as described in section 2.4. The result of this implementation was brought up on the screen and the six student participants were asked to engage in exploring the system after having received a short introduction into the interaction scheme that was conceived by the project stakeholders. The participation sessions were conducted for each participant individually at the participant's convenience. Both experimenters and submitting authors were present during each session. In each session, one of the experimenters was engaging the participant in a contextual inquiry and semi-structured interview after the participant was given time to orient him or herself and familiarize with the interface, while the other experimenter was taking notes. During the interview, questions regarding the understanding of the interaction schema of the information items on the screen were asked and the responses were recorded – this allows a backwards induction so that it can be assessed in which way the presentation of information can fuel an understanding of the presented information itself and therefore ease the interaction with the user.

In phase two, the evaluation was conducted on a Sun Ray 2 workstation connected to a central server hosting the Java applet as well as the GWT application. The Sun Ray 2 was equipped with a 22" Sun LCD display running at the native resolution of 1680x1050 pixels. Each volunteer participated in a personal session, except two professors that participated in the same session because of time constraints. Each participant was given a brief introduction as to the purpose of the interface they were currently evaluating, followed by an introduction into the specific implementation. Afterwards, they were given the opportunity to explore the interface and after some time encouraged to speak out loud about their impression and thoughts. The evaluation session therefore seamlessly moved into a contextual inquiry mode in the form of a semi-structured interview. All participants were reviewing both interfaces. The participants were introduced to a specific interface by the experimenter that was mainly responsible for implementation of the corresponding prototype. Both experimenters took extensive notes during the evaluation on the participants' responses, comments, and actions.

3.3 Results

The following sections discuss the results of the implementation evaluation. Analysis of the interaction paradigm from phase one will be outlined, as well as, the advantages and pitfalls of the interface alternatives from the second phase.

3.3.1 (Phase 1) Evaluation of Information Interaction: Static HTML

The purpose of this phase of evaluation was to assess the schema of interaction between the different information items: ABET Performance Indicators, Software Engineering Program Objectives, Courses, and Students, due to these items existing within a complex, circular interaction graph. The schema was proposed by the project stakeholder, i.e. the professors of the SUNY Oswego Computer Science department who will be tracking the ABET certification process of the Software Engineering major.

Student participants engaged in a semi-structured review of a static HTML representation of the interaction, consisting of four HTML tables on distinct HTML pages that display the interaction of items. As was expected, the participants noted that the manipulation of the information items, such as modifying mapping between items, or adding, removing, or altering information items required a direct manipulation of the HTML. It was also noted that this is an error-prone process, as it is likely that not only errors will be made when modifying the items (as no context-sensitive restriction can be provided), but also introduces the possibility of making an error when altering the HTML code. This was declared to be a tedious and cumbersome process. Furthermore, navigation between the HTML tables must be inserted manually by altering the HTML, which is also error-prone and cumbersome. Five out of six participants expressed concerns with the presentation by static HTML and one participant even mentioned the need for a dynamic visualization approach and suggested the use of a form-like interface. This suggestion has been suggested to other participants if they didn't recommend it by themselves and all of them agreed that this would be an appropriate method. The participants were naïve to this study; therefore, this outcome is very fortunate, as it was the proposed goal of the second phase.

On the positive side, all participants were able to grasp the interaction scheme of the information items with only minimal explanation. The participants were only briefed on the certification process for an IT major to receive ABET certification, and, by reviewing the way the information was represented on the screen, were able to conclude the interactions correctly. The majority of questions regarding the ABET certification process and interaction of Performance Indicators, Program Objectives, Courses, and Students were also able to be answered correctly. It can hence be concluded, that the interaction schema that was proposed by the project stakeholders is successful in modeling and visualizing the interaction of information items and thereby simplify the interaction graph.

3.3.2 (Phase 2) Prototype I: The Applet Approach

The appropriateness of the interaction schema was established in phase one, this prototype tries to visualize the information items in a form-like manner in accordance with the recommendation and agreement of the participants in phase one. When presented with the applet prototype, participants appealed to the cleanliness and form-like appearance of the interface. All participants understood that different panels represented different views of similar data, however with slightly different, more specific focus on each individual tab. A generally well-understood metaphor seemed to be the use of pull-down menus and list boxes as filter elements that allow the current view in the panel underneath the filter element to be ordered accordingly. Only one participant mentioned that the words “filter by” should be included to make it more obvious as to the purpose of the pull-down menus. A very positive aspect about the interface was the fact that the adding and modification of information items is done in a central location on a dedicated panel. Participants appreciated the cleverness of this approach, as items can be reviewed in all other panels, and quickly switch into “modification mode” by moving to the dedicated modification tab on the interface.

Another successful interface metaphor is the arrangement of detail tabs that display the mapping of different information items. The panels are aligned such that the more general information items, like Performance Indicators (i.e. information items with rather vague descriptions), are located farther to the left, whereas more specific information items (i.e. specific assessments of students' performances) are farther to the right. This arrangement therefore follows the natural left-to-right orientation that one would expect from text representations, but also introduces a hierarchy of the items according to their specificity.

A negative point about the Applet prototype was the missing of a context-sensitive help. All participants consistently asked for tool-tips or a help menu. Furthermore, the interface was considered not *verbose* enough. Feedback is necessary to explain what action was currently performed by the user (which will increase user confidence in the system) and what items can be expected to represent what. These are only superficial problems that can be addressed by simply adjusting the labeling of buttons, panels and tabs, or adding tool-tips to existing buttons and adding a status or information bar; this prototype can be considered a success.

After the participants reviewed the GWT prototype, most of them indicated that the applet prototype should be equipped with a user authentication feature that, depending on the permission level of the authenticated user, allows either to only view or to also modify items.

3.3.3 (Phase 2) Prototype II: The Web Application Approach

The GWT prototype also benefits from the positive evaluation of the interaction schema in phase one. This prototype also employs tabs to show relational data in a stream-lined fashion. While an overview screen allows for a quick glimpse of the existing items, participants critiqued the need for a “Details” panel to show the relationship of one specific item to another. It is not possible to review a collection of items and their individual relationships to other items at the same time. This was considered a mission-critical pitfall of the GWT prototype. Another issue of the GWT prototype is that – similarly to the applet prototype – tool-tips are missing from buttons, drop-down lists, radio buttons, and list boxes. This, however, is a problem with the GWT technology, as it is not possible to add such a feature to control elements of the interface. The final point of criticism is the color scheme that was used in the HTML wrapper. The prototype is fully customizable using Cascading Style Sheets however, for the purpose of this evaluation, the default color scheme was employed. This color scheme displays black text on a brown background, which was frequently experienced as distracting and hard to read due to the low contrast. One user, though, pointed out a possible usefulness of this effect: while “Detail” dialogs are modal in nature (i.e. does not allow the parent panel to be manipulated while the dialog is showing), GWT does immediately effectively communicate the modality of the dialog to the user. Hence, the color scheme could help in more effectively communicating the modality of a dialog.

A successful aspect of the GWT prototype is the Login Screen feature that comes with the ability to display certain editing-related control elements on the interface, and thereby allowing users, depending on their permission level, to modify or review the information. As the prototype automatically removes elements used to add, delete, or modify information items, users cannot be tempted to attempt an action that is not granted to them. An alternate approach pointed out by two of the participants is that a button could be “grayed out” to symbolize that this control is currently not active.

Another positive aspect of this prototype is that it is more aesthetically appealing than the Java applet. Participants appreciated the elegant design of control elements and the clean representation of information items. However, since GWT was designed to ease the development of elegant, dynamic, data driven web applications, this is not surprising.

3.4 Final Design Recommendation

From the evaluation in phase two, it became obvious that the more successful prototype is the Java applet prototype. This prototype visualizes the relational data and the information items, such as Performance Indicators, Program Objectives, Courses, and Students, significantly better than the GWT prototype, by allowing a quick overview through all existing information items and there relation to other information items. Also, the Java applet prototype is simpler for modification tasks, as adding, removing, or modifying information items is done in a central location from within a dedicated tab.

On the other hand, the GWT prototype was perceived as visually more appealing and it can therefore be assumed that this prototype will be more interesting and enjoyable to work with rather than the applet. This prototype has the aesthetics of a designed web site, and can be visually customized in its' entirety – a quality that the applet approach does not have.

As the final design recommendation, there are essentially two conclusions that can be drawn. First, the applet prototype visualizes the data more successfully by presenting different views of similar data and affording the ability to filter the items that are currently presented. This was the major advantage of the Java applet prototype. Second, the GWT prototype allows for the customization of the interfaces texts, color palette and control element layouts in a richer and simpler way. One approach to combine the advantages of both prototypes would be to implement the visualization strategy of the applet prototype in a GWT version, however with the draw back that certain interface-centric interaction features, like tool-tips, will not be available as there are currently not supported in the GWT technology.

Another recommendation for combining the positive aspects of both prototypes is to implement those security features and interface elements that were perceived as helpful in the GWT prototype into the applet. A login screen and user authentication system was the major draw back of the applet approach and was perceived as the most successful component of the GWT prototype. Hence, implementing such a feature along with more expressive and "verbose" interface components (like status bars and context sensitive help in the form of description boxes) in the applet, by taking the status representation paradigm employed in the GWT approach into account, will make the applet an ideal way of representing the complex information item interaction as described in the earlier section.

4. Conclusion and Project Review

This evaluation project was a very ambitious undertaking. Not only did it entail the implementation of two vastly different prototypes using related, but incompatible technologies, it also entailed the evaluation of these prototypes and – primarily – the interaction scheme of information items the prototypes are intended to visualize. Hence, this project was essentially two projects in one: The implementation portion and the evaluation.

The implementation turned out to be significantly more challenging than initially expected. Since the submitting authors decided to implement the underlying database schema as well as the front end prototypes of the database, a considerable amount of time had to be invested into the aspect of the database, even though it was not immediately the subject of evaluation. But since creating a mock-up of a database schema using the information items and information interaction paradigms (whose evaluation was discussed in the earlier section) was just as difficult as implementing it fully, and hence delivering a deployable product was possible as an extra benefit for the project stakeholders, the submitting authors committed to completeness rather than theoretical mock-ups.

The implementation stage of this project was also rather challenging, as technologies were selected by the authors that were not fully established to deliver the goal that was intended by the project stakeholders. While Java based technologies have in the past proven to be well-qualified to be used with a database back end for form-like information presentation, especially applets and the Google Web Toolkit reserve challenges that needed to be addressed. For instance, as both technologies are made available through the browser, security policies regarding applets disallow basic I/O operations and network connections, making it difficult to connect to remote database servers. Furthermore, this approach requires the use of database drivers, which must be available to the client without the need of setting up program dependencies or class path variables. The biggest challenge for GWT lied in the interfacing with a database as connectivity can only be achieved using asynchronous Remote Procedure Calls. Setting these up is a major undertaking when deploying the GWT application to a web server, which in turn must also be configured accordingly. Another issue was the premature state of the GWT Java API. Even though GWT is currently in version number 1.5, there are many pitfalls in the API that had to be addressed by the authors in order to still achieve the intended goal.

We believe, however, that the selection of technologies was ideal for the outcome of this project. We were able to deliver not one but two fully working, deployed and customizable prototypes that address the needs and requirements that we had established in Part II of this document. The evaluation of prototype and information interaction in Phase III shows that the gathered requirements in Phase II have accurately met the needs of the three user groups and resulted in the creation of well designed prototypes. Despite the obvious flaws in the selected technologies, the choice for these technologies was a fortunate one. Since the prototypes, especially the applet approach, achieved such high regards during the evaluation, it can be safely said that these technologies are adequate to implement the ABET certification review interface that was the focus of this project. Additionally, with this project, we have established how to address the issues of the technologies, as mentioned above. This has a benefit beyond the scope of this project, as it can serve as a guideline for future projects with similar goals. We gladly share our insights and solutions to the problems, pitfalls and challenges that we have surmounted in this project to achieve excellence. We provide our insights in Appendices B through D in the form of tutorials that address the most important issues that we have encountered. Our hope is that these tutorials will help future students and researchers to overcome problems faster so that they can benefit from the work that we have done too.

To conclude, this project was extremely successful. The authors learned a great deal in two software engineering related aspects, namely software development and product testing. We learned many important lectures in terms of requirements engineering and product evaluation as well as project time management. To wrap up the project, we have selected the more successful prototype – the Java applet – for improvement and added features that were suggested during the evaluation, like a login menu and user authentication. Furthermore, we enhanced the quality of descriptions and context-sensitive help by adding Tool Tips and modified the labeling of some control elements – all in accordance with the results of the evaluation. Our endeavor was to present a complete, ready for use system.

The database and software that has been implemented throughout the project is fully documented. The authors formally make all documentation, source code, tutorials, and implementation notes available to the project stakeholders, namely the faculty of the Computer Science Department at SUNY Oswego. It is our hope that our project software will be helpful in their work and be used frequently.

Naturally, the work doesn't stop here. Even though this project is coming to an end, future work would entail a more thorough analysis of the technological nuances of the Java applet prototype versus the GWT prototype. In this project, both prototypes use different strategies to visualize the data. Evaluation has found the strategy of the applet approach to be the more successful. However, it would be interesting to see how the strategy of the applet would perform when implemented in GWT. This question could be addressed in a future project. Future work should also address a "fine tuning" of the Java applet according to exact specifications of ABET. Since these have been rather vague, they have not been taken into account in this project. Close work with ABET could help address this question.

5. References

Brown, D. M. (2006). *Communicating Design.* San Francisco: New Riders Press.

Caroll, J. M. (2003). *HCI models, theories, and frameworks: Towards a multidisciplinary science.* San Francisco: Morgan Kauffman Publishers.

Dumas, J, & Redish, J. C. (1999). *A practical guide to usability testing.* Exeter: Intellect Ltd.

Krug, S. (2005). Don't make me think (2nd ed.). San Francisco: Morgan Kauffman Publishers.

Kuniavsky, M. (2003). *Observing the User Experience: A Practitioner's Guide to User Research.* San Francisco: Morgan Kaufman Publishers.

Niederst, J. (1999). *Web Design in a Nutshell, 1st Edition.* O'Reilly.

Sharp, Rogers, Preece. (2007). *Interaction Design: Beyond human-computer interaction (2nd Ed*). Wiley & Sons.

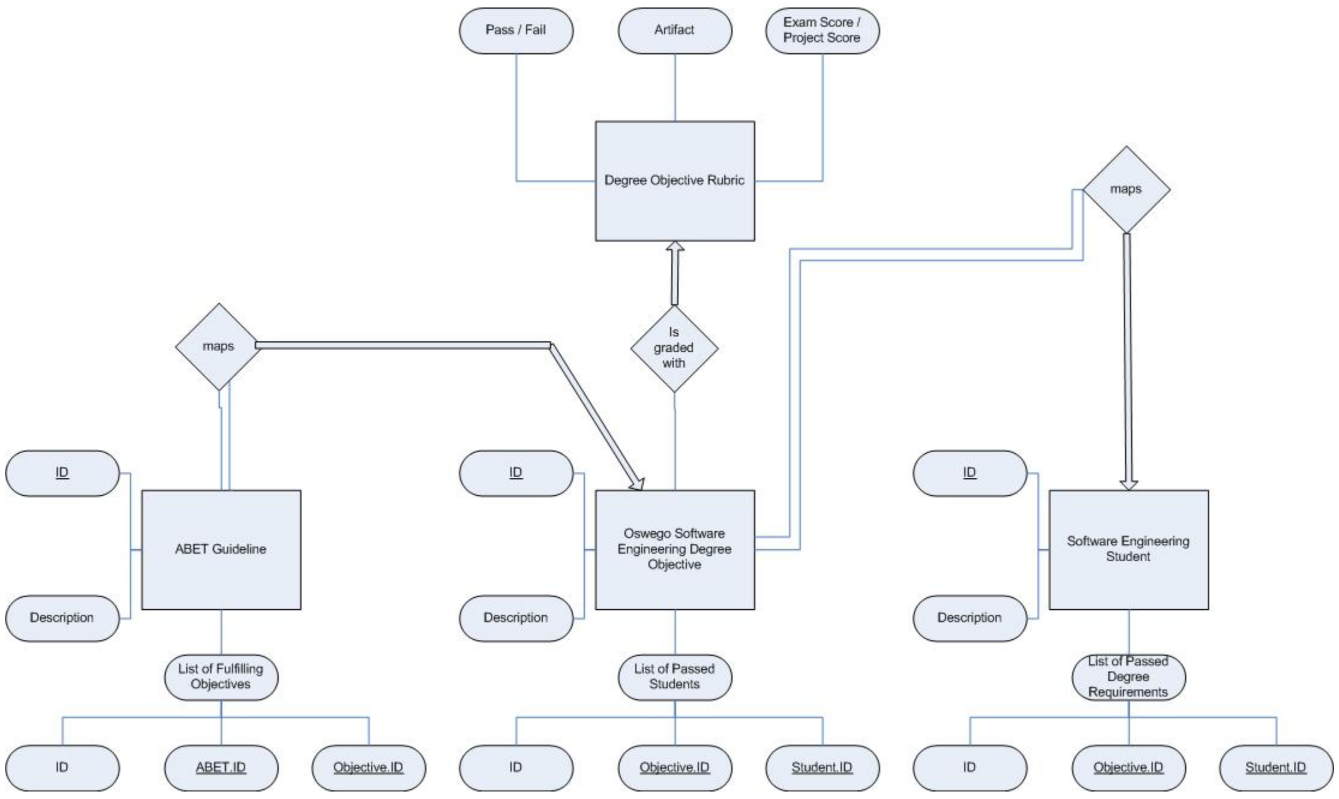
Silberschatz, A., Korth, H. F., Sudarshan, S. (2006). *Database System Concepts, 5th Edition.* McGraw-Hill Higher Education.

Rubin, J. (1994). *Handbook of Usability Testing*, John Wiley & Sons, Inc.

Appendix A – Database Normalization

This document displays the database design and discusses the notion of normalization. The aim is to point out in which – if any – normal form the database has been designed to ensure an optimal design reducing redundancy and maximizing performance. In the following, an Entity-Relationship diagram of the database can be found along with a formal definition of the relations. For each relation schema, a list of normal form constraints (given a desired normal form) that are satisfied is shown.

Entity-Relationship Diagram



Relation Schemes

In the following, please find a listing of the relation schemes of this database.

abet_guideline	= (ID, Description, objective-map)
grading_rubric	= (ID, rubric)
guideline_objective_mapping	= (ID, abet_guideline.ID, se_objective.ID)
objective_student_mapping	= (ID, Objective-ID, Student-ID, grade/passed/artifact-link)
student	= (ID, Description/Name)
sw_eng_objective	= (ID, Description, objective-student-mapping, grading_rubric.ID)

Normalization to Boyce-Codd Normal Form

From the relation schemes it follows that given an antecedent a set of precedents can be inferred. This section lists all possible conclusions that can be drawn from the antecedents and which constraint of Boyce-Codd Normal Form are satisfied. BCNF is the desired normal form for this database.

Antecedent → precedent	BCNF Constraints satisfied
abet_guideline.ID → Description, objective, Student, rubric	√ Antecedent is primary key √ Conclusion is not trivial
abet_guideline.ID, sw_eng_objective.ID → Student.ID	√ Antecedent is primary key √ Conclusion is not trivial
student.ID → grade/passed/artifact, objective, abet_guideline	√ Antecedent is primary key √ Conclusion is not trivial

All antecedents are primary keys and no precedent is trivial. It follows, that this database is in Boyce-Codd Normal Form.

Appendix B – Integrating MySQL and Java Applets

Integrating MySQL into Java Applets

A guide create signed Java Applets with Database Connectivity

Wayne Weibel, Bastian Tenbergen

wayne@wayneweibel.net| bastian@tenbergen.org

http://www.wayneweibel.net | http://www.tenbergen.org

Human-Computer Interaction MA Program

Department of Psychology

Department of Computer Science

State University of New York, College at Oswego

Oswego, NY, USA

Introduction.

Why this tutorial? As you may or may not know, Java Applets run in a so-called sandbox – a simplified environment in the browser that makes executed foreign code safe and harmless. This sandbox has a very strict security policy and does not allow Java Applets to perform Input/Output operations such as reading and writing files to disk or utilizing the network interface on the client machine. Yet, it is possible to do so, but it requires signing the JAR file, the Applet lives in. A different common issue with Web distributed JAR content is the correct set-up of class paths and dependencies. Information on this topic are rather sparse and not beginner-friendly. This tutorial suggests a work-around, which may work for many projects.

Assumptions.

I will assume that you have an understanding of how to do create Java Applets in first place. I will assume that you have read other tutorials (which you should find by google'ing [4]) that explain to you how Applets differ from applications and how to implement them. You will most likely want to try and develop your Applet within Eclipse, so I will assume that you have done just that and everything is working just fine. Another assumption is that you have access to some web server, and that you can use the world's largest knowledge base [4] to find out how to deploy Applets to an HTML document. Lastly, I will assume that you are familiar with manipulating Zip files – either using some graphical tool or the command-line.

Integrating MySQL into Java Applets.

In this little tutorial, we will deploy a simple MySQL enhanced Java Applet to an HTML page. This is universally, completely independent from your application, and only depends on the Applet code itself. Therefore, we will keep this tutorial as generic as possible.

1. The Short Version. Like always, this is for the impatient.

Do the following steps and it will work (assuming your Applet is running just fine in Java's Applet viewer and you are done developing for now):

- Remove all MySQL driver resources from the class path Yes, you heard right.
- Create a JAR file for your Applet.
- Copy the MySQL Driver binaries from the MySQL Driver JAR into the root folder of the Applet JAR.
- Create a certificate in you local keystore.
- Export the certificate.
- Sign the Applet JAR that contains the MySQL Driver binaries.
  The certificate will be valid for 6 months only.
- Deploy the Applet JAR to a HTML page hosted on a web server.
- Stir, serve, and enjoy - you're done :-)

Now for the detailed stuff.

2. Necessary Software.

Of course, you will need the MySQL Driver binaries for Java [7]. Install it, by adding it to your class path I strongly recommend to use some form of Integrated Development Environment (IDE) for your development. Develop your Applet and test it in the IDE's (ideally, Eclipse [4], version 3.4 or higher, also known as Ganymede [5]) hosted mode using Java's Applet viewer You will also need a database you can use - get WAMP [6], as it comes with all you need for that. MySQL, a good free database server and some PHP-based management software. Make sure MySQL in WAMP is running when you deploy. Your MySQL server must be reachable from the Internet to be of any value for you in production. You will also need a Zip Utility, like WinZip, PowerZip, WinRar or whatever. *NIX flavor operating systems typically come with a cool command-line tool that you ought to be familiar with.

3. Remove Driver from Class path

This step is optional. Since you already have written your Applet, and all you want to do is deploy it, I think it is safe to assume that you have your MySQL Driver set up in your project's class path This may be either a global one, being valid for all your projects or even your entire development machine, or this may be just for this one project. The latter method is how Eclipse likes to do it. To make things easier when you create the JAR file, you may want to remove the MySQL Driver JAR from the class path In Eclipse, simply right-click on your project, navigate to `Properties` in the context-menu and find `Java Built` on the left hand side. Access that menu and follow the screen instructions to remove the JAR from your project class path

4 Create Applet JAR File.

Now you are ready to build the JAR file that will host your Applet. Certainly, one can run Applets as a collection of class files in a folder structure from a web server, but this won't allow you to sign the JAR. Right-click on your project in Eclipse, then go to `Export...`. Under `Java`, find the entry `JAR file...` and follow the instructions of the wizard. You may select whether or not you would like to include source files at your convenience, but when it asks you if you would like to include dependencies and resources, make sure to **not specify** the MySQL Driver – we will do this in the next step. Continue with the wizard. At the appropriate step, it will ask you if you would like to provide a Manifest file. Make sure to advise the wizard to auto-generate one on your behalf. This is necessary, as otherwise signing the JAR later on might fail. You may close your IDE now – we won't need it anymore.

5 Copy MySQL binaries into Applet JAR.

Now for a somewhat unusual part – we are placing all binaries from a resource JAR into the new Applet JAR. The reason why we didn't do this in step 6.4, but are doing it independently is that Eclipse sometimes tries to de-compile class files, when it kind find the source, and this might result in problems with MySQL. Also, doing it the other way, i.e. finding MySQL Driver source files and compiling it yourself is quite an undertaking, which is unlikely to succeed – therefore, we do it the manual way.

In the file system of your development computer, find the MySQL Driver binary JAR that you once downloaded and the Apple JAR that you have just created. Open both in your favorite Zip utility, I personally prefer WinRar, or the command-line. In the MySQL Driver JAR, you will find two directories – `org` and `com`. Copy both of them into the Applet JAR into the root folder. Alternatively, you may unzip the MySQL Driver JAR and add the contents to your Applet JAR. Careful, however! Make sure not to overwrite the Manifest file if you use this method!

6 Create Certificate in local Keystore.

The directions in this section are based on a forum entry (http://forums.sun.com/thread.jspa?threadID=174214 , accessed December 2008). This method is one of the few universally working methods that I discovered and it is fairly easy.

Open a command-line window. Enter the command

```
keytool -genkey -keyalg rsa -alias yourKey
```

where `yourKey` is some alias under which you can refer to the key you will now create. Naturally, you may specify any key algorithm using the `-keyalg` argument, but RSA will work just fine and is as good as any. Answer the questions the tool will ask you for – don't worry. In the end, you may review the answers and make changes. Remember to remember your password well! Especially the one for the Keystore, as this is the master password for all keys.

7 Export Certificate.

This step is also optional. Exporting the key is however quite useful when you want to sign many things with the same key on different machines. In that case, you have to import the file you exported after this step (we will not cover that in this tutorial, though).

Still in command-line mode, enter the following command:

```
keytool -export -alias yourKey -file file.ext
```

where `yourKey` is the key alias you have specified in step 6.6 and `file.ext` is some file name with some extension. A convention is to use the alias of the key as the file name with the extension `.crt`. Hence, if your alias is `yourKey`, it is a good idea to call the file `yourKey.crt`.

When you hit enter, the key with the specified alias will be placed in the certificate file in the current working directory. You may have to move it into the same directory of your Applet JAR.

8 Sign the Applet JAR.

Now for the step that makes it all possible to run Applets with I/O permissions and MySQL: We are signing the Applet JAR. A word on this, however. In this tutorial, we use what is called a self-signed certificate. These may or may not be acceptable for every user, depending on their individual (or corporate) security policies regarding foreign interpreted code. After all, the code you have written could cause harm. Also, **the certificate will be valid for only 6 months** by default, which means that after 6 months, your users will receive a message, indicating that the certificate of the Applet has expired. When that happens, you will need to repeat this step again and re-sign your JAR.

To sign your Applet and MySQL Driver JAR, simply enter the following in command-line:

```
jarsigner Applet.jar yourKey
```

`Applet.jar` is the name of the Applet JAR that you have created in step 6.4 and that contains the MySQL Driver binaries from step 6.5. Note, that you will have to specify the same key from step 6.6 for what is called `yourKey` in this example. Also, you will be prompted for the Keystore password – I hope you still remember it. After this is done, you may verify the JAR:

```
jarsigner -verify -verbose -certs Applet.jar
```

And it should tell you some details about the JAR and let you know if an error is present.

9 Deploy Applet JAR to HTML Document on Web Server.

You are done. Well almost. You Applet is now ready to be run from within a web browser and for that, you of course need to upload your Applet to a server and integrate it to an HTML Document. How this is done is simple and easy and tutorials can be found plenty on the Internet. A short quick start shall be this, just for completeness:

Create an HTML Document and add the `<applet>` tag. Enter the main class of your Applet that extends `java.applet.Applet` in the `code` attribute and naturally the Applet JAR in the `archive` attribute. Somewhat like this:

```
<applet code="com.yourdomain.packagename.YourAppletWithMySQL.class"
                        name="Some name goes here"
                        archive="AppletWithMySQL.jar"
                        width="1000" height="700">
                        Your browser is not Java enabled.
</applet>
```

Summary.

Well, this is it. Your Applet is now available through the HTML Document on your web server and should work with your MySQL database. I hope you like this little tutorial and find it helpful in your next Applet programming venture. Let me know of any successes, failures, and whatnot.

As the other GWT tutorial, this document was made possible, in part, by many kind souls in various forums across the Internet that post answers to questions of others. Those people are way to many to mention them all, but all of them shall be thanked much.

References.

[1] Dewsbury, R. (2007). *Google Web Toolkit Applications.* Boston, MA: Addison-Wesley Professional.

[2] Geary, D. with Gordon, R. (2008). *Google Web Toolkit Solutions.* Boston, MA: Prentice Hall PTR.

[3] Google Search Engine, accessed October, 24th, 2008

http://www.google.com/

[4] The Eclipse Project, accessed October, 24th, 2008

http://www.eclipse.org/

[5] Eclipse IDE v3.4 “Ganymede”, accessed October, 24th, 2008

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip

[6] WAMPServer – Apache, MySQL, PHP on Windows, accessed October, 24th, 2008

http://www.wampserver.com/en/

[7] MySQL Connector/J, accessed October, 24th, 2008

http://www.mysql.com/products/connector/j/

[8] Niederst, J. (1999). *Web Design in a Nutshell, 1st Edition.* O'Reilly.

[9] Silberschatz, A., Korth, H. F., Sudarshan, S. (2006). *Database System Concepts, 5th Edition.* McGraw-Hill Higher Education.

Appendix C – Database Access in GWT – The Missing Tutorial

Database Access in GWT – The Missing Tutorial

A programmer's guide on integrating MySQL into Google Web Toolkit Applications using the Eclipse Programming Environment

Wayne Weibel, Bastian Tenbergen

wayne@wayneweibel.net | bastian@tenbergen.org

http://www.wayneweibel.net | http://www.tenbergen.org

Human-Computer Interaction MA Program

Department of Psychology

Department of Computer Science

State University of New York, College at Oswego

Oswego, NY, USA

Introduction.

Why this tutorial? Isn't there enough info on this on the Intertubes? Well, yes. And no. There are a few good sources, especially the ones from the makers [3], but none really explains the special nuances, like that the DB driver has to live in a Servlet. In fact, that is the most important take-home message: The actual database queries are handled by a **Servlet** and you need to make **RPC calls**, using a **callback handler** to have the client side GUI (recall that the GWT application is executed client side in the browser, not on the server!) talk to the database. The mission of this tutorial is hence to show the necessary steps needed to integrate MySQL with GWT applications and explain what happens at each step.

Assumptions.

I will assume that you have an understanding of how to do GWT in first place. At least you should know that GWT is a Java front-end for dynamic Javascript/AJAX Web Applications, that you the main class implements the EntryPoint interface and all the other stuff to get a very basic GWT App working in your browser. I will also assume that you know basic Java, like class paths, dependencies and such. I also will assume that you have some dynamic web server, like Tomcat, at your disposal, and that you can use the world's largest knowledge base [4] to find out how to configure it.

The Missing Tutorial.

In this little tutorial, we will implement a simple thing – a LoginScreen that will allow you to enter a user name and a password, and authenticate this combination. Assuming, of course, you have a table in the database that lists a user name, with a password. Let's just assume your GWT application has just this one class, LoginScreen, that implements EntryPoint and hence has an public void onModuleLoad() method.

1. The Short Version. For the impatient.

Do the following steps and it will work (assuming your GWT is up and running and all you have to do is implement the DB connection):

1. Add your DB connector JAR to your Tomcat's class path
2. Add **TWO** interfaces to your project's client package that specify all the methods you can call from within your GWT user interface: One for synchronous callback, one for asynchronous callback. In those, specify the methods that will provide DB queries for you.
3. Add a DB connection class to your project's server package that will handle all the DB queries, just like with any other Java application. This class must extend RemoteServlet and implement your synchronous interface from point 2.
4. Add a RPC provider that implements the asynchronous interface to the GWT application component that needs to query the DB. Also add a nested class to that GUI component that implements AsyncCallback. This handler will do stuff with your GUI depending on the query result.
5. Now you can call the method specified by your interfaces using the RPC provider and applying the callback handler.
6. Create some serializable objects on your client side - in case you want to send more than primitive types (including Strings) to and from the database.
7. Stir, serve, and enjoy - you're done :-)

Now for the detailed stuff.

2. Necessary Software.

Of course, you will need GWT [5]. Download it [6] and install it, of course. I strongly recommend to use some form of Integrated Development Environment (IDE) for your development. In fact, without one, you are kind of screwed. Get yourself Eclipse [7], version 3.4 or higher, also known as Ganymede [8]. Make sure to get the Java Enterprise version of Eclipse, as this comes with all the goodies for Web Programming - even with said dynamic Web Servers. The easiest way to develop GWT stuff using Eclipse is getting a GWT focused plug-in. I recommend Cypal Studio [9] - it is free, and the documentation is just what you need, even though the docs [10] are somewhat sparse. Follow their how-to on getting started integrating Cypal Studio and making your first GWT App. You will also need a database you can use - get WAMP [11], as it comes with all you need for that. MySQL, a good free database server and some PHP-based management software. Make sure MySQL in WAMP is running while you develop. Of course, you can use some remote MySQL server, too, like the one your ISP provides you with.

3. Add MySQL to Java.

Next, let's take a look on what you have to do to get everything done right. You need to make sure that you have your database driver - get the MySQL Java DB Connector [12] that fits to the MySQL version that came with your WAMP. This should be a JAR file, which you have to add to your class path **Careful now!** You need to add it to your Eclipse class path *and* your Tomcat class path! Otherwise Eclipse won't let you access the MySQL API and/or Tomcat can't find the DB driver later on. Here is how to do it: Remember that Eclipse specifies a class path for each project individually. Right-click on your project, click on properties, then on Java Build Path. Click on Add External Jar... and find the MySQL DB Connector JAR file to the project from your file system. This will allow you to have access to the MySQL Java Connector API from within Eclipse - this way the auto completion and syntax checking and all that nice IDE stuff works. While you are at it, in the same window, make sure Cypal Studio added the file "gwt-servlet.jar" and "gwt-user.jar" libraries to the build path of your project - for the same reason ;-) Next, check that the WebContent folder in your project has the same jar added to it. Do so by expanding the WebContent folder, navigate to WEB-INF, then lib. Into there, you can just drag-and-drop the MySQL Connector JAR file from your file system - Eclipse will do the behind the scenes adding yourself.

4. Add MySQL configuration to GWT.

Now, you have to tell the myriad of XML configuration files of GWT that you have, in fact, a Servlet that supports MySQL. Well, you will have one, after the next step. Under "Java Resources: src" is all your Java configuration and source file stuff. Navigate to the folder of your project that is named after the package that all your classes are in. This should be the first one down from the "Java Resources: src" folder and look something like `com.yourdomain.projectname` or whatever you have chosen as your project package. In there, you will find not only the client and server sub packages, but also a folder that contains a sub folder public and an XML configuration file, named after your main class. Remember, that in your GWT application, you need some class that implements the `EntryPoint` interface from the `com.google.gwt.core.client` package. We said earlier that this class is `LoginScreen`, so the configuration file you want to look for is `LoginScreen.gwt.xml`. Add the following line to this configuration point:

```
<servlet
class="com.yourdomain.projectname.server.MySQLConnection"
path="/MySQLConnection" />
```

The class's package name must be the one you have set for your project. Server is a sub package that is there anyways, because GWT wants it. MySQLConnection may be called something else - this is a class you will write later on. The path argument is rather is arbitrary, but must match up with the RPC initialization inside Java class. More about that later, too.
In the end, you should have some form of configuration XML file like this:
LoginScreen.gwt.xml:

```
<module>
```

```
        <!-- Inherit the core Web Toolkit stuff. -->
        <inherits name='com.google.gwt.user.User'/>
        <!-- Specify the app entry point class.  -->
        <entry-point class='com.yourdomain.projectname.client.LoginScreen'/>
    <!-- servlet context - path is arbitrary, but must match up with the rpc
init inside java class -->
    <!-- Tomcat will listen for this from the server and waits for rpc
request in this context -->
        <servlet class="com.yourdomain.projectname.server.MySQLConnection"
path="/MySQLConnection" />
        <inherits name="com.google.gwt.user.theme.standard.Standard"/>
        <inherits name="com.google.gwt.user.theme.chrome.Chrome"/>
        <inherits name="com.google.gwt.user.theme.dark.Dark"/>
</module>
```

5. Clean your project.

This will make sure that all your configurations are set-up correctly, re-configure your web server and compile the GWT project so that it will run properly again. Do so, by clicking in Eclipse's menu bar on Project, then Clean... and select your project. It will clean, compile, and prepare for execution, potentially add missing configurations and repairing ambiguities. You may need to help out a little bit, when Eclipse asks you, but that should be trivial.

6. Make the client side be aware of the Servlet.

Now, we need to make some client side preparations for stuff to happen. You need to create **TWO** interfaces. Both must live in the `com.yourdomain.projectname.client` package. You are allowed to put it in a sub package, but it must be in client, or the Ninjas will get you. Now, as I said, two interfaces: one for synchronous callback, one for asynchronous callback. Let's say, you call the synchronous interface `DBConnection`. This interface **must** extend the `com.google.gwt.user.client.rpc.RemoteService` interface. In this interface, you may specify all the methods you'd like to use to connect to the database. Let it be one big fat method, that's fine. But I recommend to make one method for every specific update, insert, delete, whatever... this is the better way, I think. You can give it any return type, but for non-primitive types, you will need to create serializable objects. More about that later. Now, ideally, Cypal Studio will have created the asynchronous interface for you and call it `DBConnectionAsync`. As you can see, the asynchronous interface specifies the same methods you have written into the synchronous interface, with two differences: The return type is always void and every method has an additional parameter: an `com.google.gwt.user.client.rpc.AsyncCallback` object. `AsyncCallback` is in fact an interface, but - again - more on that later. Every `AsyncCallback` parameter is equipped with Java Generics, the generic type being the return type of the method in the synchronous version. Don't try to change it - in fact, you need to make sure that what I just explained in fact is automatically done by Cypal Studio. So go ahead, double-check, that you have two interfaces that are set up in the described way. Done? Good. You hence should have two files like this:

DBConnection.java:

```
public interface DBConnection extends RemoteService {
     public User authenticateUser(String user, String pass);
}
```

DBConnectionAsync.java:

```
public interface DBConnectionAsync {
     public void authenticateUser(String user, String pass,
AsyncCallback<User> callback);
}
```

Note the User object as return value... more on that in the next paragraph. Of course you can - instead of a User object - also just return a Boolean if authentication was successful - your choice. Depends on how you want to use your authentication, I guess.

7. Create serializable objects for callbacks.

Also on the client side (i.e. in the `client` package), create one object for every non-primitive return value that you have depicted in the methods in your interfaces. You don't have to do create objects for primitive types, like Boolean, Integer, Double, or anything in the `java.lang`, or `java.util` packages. GWT will do that for you and simply use Javascript equivalents. You can even use Arrays and `java.util.Vector` with Generics. The only thing you can't use is `java.lang.Thread`, for obvious reasons (wanna know those reasons? This shall be a thought excerise for you. Hint: You can't serialize Threads in plain vanilla Java anyways. You know why?)

GWT is VERY picky [13] with what is serializable and what not, as GWT's idea of serializing an object is significantly different from Java's (and pretty much everyone else's) way of doing so. A class in GWT is serializable, if (according to the website [13]) **one** of the following holds true:

- is primitive, such as `char`, `byte`, `short`, `int`, `long`, `boolean`, `float`, or `double`;
- is `String`, `Date`, or a primitive wrapper such as `Character`, `Byte`, `Short`, `Integer`, `Long`, `Boolean`, `Float`, or `Double`;
- is an array of serializable types (including other serializable arrays);
- is a serializable user-defined class; or
- has at least one serializable subclass

A user-defined class is serializable, if **all** of the following are true (according to the website [13], again):

1. it is assignable to IsSerializable or Serializable, either because it directly implements one of these interfaces or because it derives from a superclass that does
2. all non-`final`, non-`transient` instance fields are themselves serializable, and
3. it has a `public` default (zero argument) constructor

The objects you create must hence fulfill the above requirements. Here is an example object that we could use by our authenticateUser method instead of a Boolean return value:

User.java:

```
import com.google.gwt.user.client.rpc.IsSerializable;
public class User implements IsSerializable {
    private String username;
    private String password;
    @SuppressWarnings("unused")
    private User() {
        //just here because GWT wants it.
    }
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
}
```

Note, how the zero parameter constructor is present, but private. This is just to illustrate that the constructor MUST be there, but not necessarily be used. You can hide it by making it private. I used the @SuppressWarnings("unused") Java annotation to make the Java compiler shut up about unused private method elements in the class... apparently javac doesn't know about GWT (surprise...). You don't have to do this, though, but it is good code.

8. Create the Servlet.

Finally, you get to create what's actually important for the DB queries. Up to now, all we have done was some "pluming" around GWT, Java, and your Tomcat to make what comes now as simple and smooth as possible. On your server package, i.e. `com.yourdomain.projectname.server`, create a class called `MySQLConnection`. Class name and package **must** be identical to what you have specified in the class argument in your `LoginScreen.gwt.xml` configuration file. Remember that one? Double-check, that class name and package name is correct.
This `MySQLConnection` class will do all the DB work for you. As I said, it is a Servlet, hence it must extend the `com.google.gwt.user.server.rpc.RemoteServiceServlet` class. Additionally, it **absolutely must** implement the synchronous interface from your client side. Hence, you will write your queries for everything you want to do with your database right in this class. Here is a full and good implementation of the class:

MySQLConnection.java:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Vector;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import com.yourdomain.projectname.client.User;
public class MySQLConnection extends RemoteServiceServlet implements
DBConnection {
        private Connection conn = null;
        private String status;
        private String url  = "jdbc:mysql://yourDBserver/yourDBname";
        private String user = "DBuser";
        private String pass = "DBpass";
        public MySQLConnection() {
                try {
                        Class.forName("com.mysql.jdbc.Driver").newInstance();
                        conn = DriverManager.getConnection(url, user, pass);
                } catch (Exception e) {
                        //NEVER catch exceptions like this
                }
        }
        public int authenticateUser(String user, String pass) {
                User user;
                try {
                        PreparedStatement ps = conn.prepareStatement(
                                        "select readonly * from users where
username = \"" + user + "\" AND " +
                                        "password = \"" + pass + "\""
                        );
                        ResultSet result = ps.executeQuery();
                        while (result.next()) {
                                user = new User(result.getString(1),
result.getString(2));
                        }
                        result.close();
                        ps.close();
                } catch (SQLException sqle) {
                        //do stuff on fail
                }
                return user;
        }
}
```

Note a couple of things here. Firstly, **NEVER, EVER,** catch exceptions the way I do in here... by catching a generic Exception object. That's bad code and not only the Ninjas, but also the Pirates will get you. Secondly, I included a detailed list of import statements so that you can see what is actually made use of. Note the User object - this is the serializable object you created on client side. The URL to the database can be anything, but must be the URL or your MySQL server. Could be localhost or your ISP. Who knows. The username and password here are NOT what you are trying to authenticate in your GWT application. In this case, the username and password are those that your ISP has given you to connect to your database. I assume that you will have some table called user in your DB with the columns "user" and "password" and we hence query the database to

`SELECT * FROM user WHERE user = 'your authentication user name' AND password = 'your authentication password'`.

That's it! After that, just do stuff with the result, as in create a User object (the seriablizable one from the client package of your GWT app) and return it.

Let's see what we have done so far. We have:

- added MySQL Connector/J to Eclipse and Tomcat
- Configured the GWT project XML file to be aware of a DB connector Servlet
- Created the synchronous and asynchronous interface
- Created the serializable database objects needed for callback
- Created the Servlet that will query the actual database.

Not bad at all. What's next is making your client side GWT GUI issue the RPC and do stuff with the result.

9. Prepare an RPC Provider in your GWT App.

Of course, all this pluming and coding is worth nothing if you have nothing using it. And since you want to feed your database output into the GUI and do stuff with it accordingly, you will need to make your GUI component call the database. For that, you need to create an RPC provider. I typically do this once when the application gets loaded, as the RPC call provider may be shared across all database accessing GUI components. In your `EntryPoint` class - `LoginScreen.java` - add a variable of type `DBConnectionAsync`. Since `DBConnectionAsync` is an interface, you will not be able to instantiate it directly, but that doesn't matter - you will just ask GWT to factor a method for you. This is how it may look like:

LoginScreen.java:

```
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;
import com.google.gwt.user.client.ui.RootPanel;
public class LoginScreen implements EntryPoint {

        private DBConnectionAsync rpc;

        public LoginScreen() {
          rpc = (DBConnectionAsync) GWT.create(DBConnection.class);
        ServiceDefTarget target = (ServiceDefTarget) rpc;

        // The path 'MySQLConnection' is determined in
./public/LoginScreen.gwt.xml
        // This path directs Tomcat to listen for this context on the server
side,
        // thus intercepting the rpc requests.
        String moduleRelativeURL = GWT.getModuleBaseURL() +
"MySQLConnection";
        target.setServiceEntryPoint(moduleRelativeURL);
        }
        public void onModuleLoad() {
            RootPanel.get().add(this);
            }
        }
}
```

The factoring mechanism is similar to what you do in the DB Driver instantiation: You ask the static method create in the GWT class to return an implementation of your `DBConnection` interface. Conveniently, the only known implementation is your `MySQLConnection` Servlet, which is hence stuffed into your `DBConnectionAsync` variable called "rpc" and will hence act as your Remote Procedure Call Provider. Next, you need to set the service entry point to the path of your connection provider. Remember in your `LoginScreen.gwt.xml` file? You have specified a path argument in your `servlet` tag. The path URL that you are providing here **must** be the one you have provided in the XML file, otherwise things will fail.

10. Make Remote Procedure Calls from your GWT application.

Now, all you have left to do is create your actual GUI in this `EntryPoint` class, preferably with some buttons, and upon clicking on of these buttons, do something. This "something" is in fact issuing the RPC call with the call provider. Of course, you will need an callback handler that takes the result and does something with it. What I always do to accomplish that is create a nested, private class that implements the `com.google.gwt.user.client.rpc.AsyncCallback` interface, with appropriate Generic type. This interface specifies an `onSuccess` and `onFailure` method, depending on whether or not the RPC call failed. Here is how it could look like:

nested, private AuthenticationHandler class:

```
private class AuthenticationHandler<T> implements AsyncCallback<User> {

     public void onFailure(Throwable ex) {

          RootPanel.get().add(new HTML("RPC call failed. :-(");
     }

     public void onSuccess(User result) {

          //do stuff on success with GUI, like load the next GUI element

     }

}
```

Note the use of Generics to retrieve the correct result type. The User object that is used here is the same, serializable object that you have created much earlier.
Now, all you have to do is call the RPC provider. Let's assume you have a button called OK, you will need a `ClickListener` and issue the RPC call when OK is clicked. The RPC call needs an instantiation of the `AsyncCallback` class you have just written. Like this:
OK button ClickListener:

```
public void onClick(Widget sender) {
     if (sender.equals(OK)) {
          AsynchCallback<User> callback = new AuthenticationHandler<User>();
          rpc.authenticateUser(username, password, callback);
     }
}
```

That's it! Essentially, you are done at this point! Below is an example implementation of how the entire, RPC enabled, callback handling, DB query issuing `LoginScreen` class may look like, with GUI components and all:

complete LoginScreen.java:

```
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;
import com.google.gwt.user.client.ui.*;
public class LoginScreen implements EntryPoint, ClickListener {

        private DBConnectionAsync rpc;
          private TextBox usernameBox;
          private TextBox passwordBox;
          private Button OK;

        public LoginScreen() {
          rpc = (DBConnectionAsync) GWT.create(DBConnection.class);
        ServiceDefTarget target = (ServiceDefTarget) rpc;

        // The path 'MySQLConnection' is determined in
./public/LoginScreen.gwt.xml
        // This path directs Tomcat to listen for this context on the server
side,
        // thus intercepting the rpc requests.
        String moduleRelativeURL = GWT.getModuleBaseURL() +
"MySQLConnection";
        target.setServiceEntryPoint(moduleRelativeURL);
           initGUI();
        }
        public void onModuleLoad() {
            RootPanel.get().add(this);
            }
        }
          private void initGUI() {
                Grid g = new Grid(3, 2);
            usernameBox = new TextBox();
                passwordBox = new TextBox();
                OK = new Button(„OK");
                g.setWidget(0, 0, new Label(„Username: „));
                g.setWidget(0, 1, usernameBox);
                g.setWidget(1, 0, new Label(„Password: „));
                g.setWidget(1, 1, passwordBox);
                g.setWidget(2, 1, OK);
          }
          public void onClick(Widget sender) {
                if (sender.equals(OK)) {
                  AsyncCallback<User> callback = new
AuthenticationHandler<User>();
                       rpc.authenticateUser(usernameBox.getText(),
passwordBox.getText(),
                                                   callback);
                }
        }
        private class AuthenticationHandler<T> implements AsyncCallback<User>
{
             public void onFailure(Throwable ex) {
            RootPanel.get().add(new HTML("RPC call failed. :-(");
                }
             public void onSuccess(User result) {
            //do stuff on success with GUI, like load the next GUI element
             }
        }
}
```

Summary.

OK, that's it! This completes the tutorial. If we did everything right, it should work just fine. It took me forever to write this stuff up, so, I'd appreciate any comments you may have. And it took me even longer to figure out how to do all this... I didn't have any help like you do :-p If you have questions, don't hesitate to post, or send me a mail.

Hope you like this little tutorial and find it helpful in your next GWT programming venture. Let me know of any successes, failures, and whatnot.

9. References.

[1] Dewsbury, R. (2007). *Google Web Toolkit Applications.* Boston, MA: Addison-Wesley Professional.

[2] Geary, D. with Gordon, R. (2008). *Google Web Toolkit Solutions.* Boston, MA: Prentice Hall PTR.

[3] Google's GWT MySQL Example Project, accessed October, 24th, 2008
http://code.google.com/p/gwt-examples/wiki/project_MySQLConn

[4] Google Search Engine, accessed October, 24th, 2008

http://www.google.com/

[5] GWT – Google Web Toolkit, accessed October, 24th, 2008

http://code.google.com/webtoolkit/

[6] GWT Download Section, accessed October, 24th, 2008

http://code.google.com/webtoolkit/download.html

[7] The Eclipse Project, accessed October, 24th, 2008

http://www.eclipse.org/

[8] Eclipse IDE v3.4 "Ganymede", accessed October, 24th, 2008

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip

[9] Cypal Studio, GWT Plug-in for Eclipse Ganymede, accessed October, 24th, 2008

http://www.cypal.in/studio

[10] Cypal Studio Documentation, accessed October, 24th, 2008

http://www.cypal.in/studiodocs

[11] WAMPServer – Apache, MySQL, PHP on Windows, accessed October, 24th, 2008

http://www.wampserver.com/en/

[12] MySQL Connector/J, accessed October, 24th, 2008

http://www.mysql.com/products/connector/j/

[13] GWT Serializable Types, accessed October, 24th, 2008

http://www.gwtapps.com/doc/html/com.google.gwt.doc.DeveloperGuide.RemoteProcedureCalls.SerializableTypes.html

[14] Niederst, J. (1999). *Web Design in a Nutshell, 1st Edition.* O'Reilly.

[15] Silberschatz, A., Korth, H. F., Sudarshan, S. (2006). *Database System Concepts, 5th Edition.* McGraw-Hill Higher Education.

Appendix D – Deploying MySQL Enhanced GWT Applications

Deploying MySQL Enhanced GWT Applications

A guide on deploying GWT Applications with MySQL Servlets to live dynamic web servers.

Wayne Weibel, Bastian Tenbergen

wayne@wayneweibel.net | bastian@tenbergen.org

http://www.wayneweibel.net | http://www.tenbergen.org

Human-Computer Interaction MA Program

Department of Psychology

Department of Computer Science

State University of New York, College at Oswego

Oswego, NY, USA

Introduction.

Why this tutorial? Isn't there enough info on this on the Intertubes? Well, yes. And no. Just like the first tutorial on GWT that I have written, this document aims at compiling all available information, getting rid of any contradictions and false information, and serving as one resource detailing the simplest possible way.

Recall: The actual database queries are handled by a **Servlet** and you need to make **RPC calls**, using a **callback handler** to have the client side GUI (as the GWT application is executed client side in the browser, not on the server!) talk to the database. The mission of this tutorial is hence to show the necessary steps needed to deploy MySQL enhanced GWT applications and explain what happens at each step.

Assumptions.

I will assume that you have an understanding of how to do GWT in first place. I will assume that you have read the other tutorial (which you should find at the same location you found this tutorial, or at least by google'ing [4]). You will most likely want to try and develop your GWT app in hosted mode within Eclipse, using Cypal Studio, so I will assume that you have done just that and everything is working just fine. Lastly, the assumption is that you have some dynamic web server, like Tomcat, at your disposal, and that you can use the world's largest knowledge base [4] to find out how to configure it.

Deploying MySQL enhanced GWT applications.

In this little tutorial, we will deploy a simple GWT application that uses a MySQL database connection that you have created using Eclipse and Cypal Studio. This is universally, completely independent from your application, and only depends on the Servlet, really. But for the more hands-on parts, we will take the little `LoginScreen` app from the other tutorial and it's `MySQLConnection` Servlet as the living example.

1. The Short Version. Like before, this is for the impatient.

Do the following steps and it will work (assuming your GWT application is running just fine in hosted mode and you are done developing for now):

1. Add your Servlet information to the `web.xml` file.
2. Create a WAR file containing your project and any resources.
3. Copy the MySQL Driver to the `$TOMCAT_HOME/common/lib` folder of your Tomcat.
4. Copy the WAR file from step 2 into the `$TOMCAT_HOME/webapps` folder of your Tomcat.
5. Start your Tomcat and watch it automatically deploy your GWT application.
6. Access your application via `http://servername:port/warfilename/`
   Where `servername` is the URL of your server, `port` the port you told Tomcat to run on and `warfilename` is the file name of the WAR file you have copied into Tomcat's `webapps` folder.
7. In the hence displayed file listing, look for the HTML file that contains is named after the Java class that implements the `EntryPoint` interface, in this case `LoginScreen.html` and click on it.
8. Stir, serve, and enjoy - you're done :-)

Now for the detailed stuff.

2. Necessary Software.

Of course, you will need GWT [5]. Download it [6] and install it, of course. I strongly recommend to use some form of Integrated Development Environment (IDE) for your development. Develop your Web app and test it in the IDE's (ideally, Eclipse [7], version 3.4 or higher, also known as Ganymede [8]) hosted mode using the integrated dynamic Web Server (most likely Tomcat Lite). I recommend Cypal Studio [9] for GWT integration with Eclipse, as it is free, and the documentation is just what you need, even though the docs [10] are somewhat sparse. Follow their how-to on getting started integrating Cypal Studio and making your first GWT App. You will also need a database you can use - get WAMP [11], as it comes with all you need for that. MySQL, a good free database server and some PHP-based management software. Make sure MySQL in WAMP is running when you deploy. Get yourself a dynamic web server, ideally Tomcat 5.5 [16] or higher. Of course, your Tomcat and MySQL servers must be reachable from the Internet to be of any value for you in production.

3. Modify Web.xml for Servlet.

Cypal Studio will already have created the necessary folder structure and file system for your GWT application project. That's why you should follow the instructions on the GWT Documentation Page [10] or my other tutorial on GWT. In the folder structure, navigate to the `WebContent/WEB-INF` folder. In there, locate the file web.xml and add the following information to the end, before the document closing tag:

Web.xml:

```
<servlet>
        <servlet-name>
                MySQLConnection
        </servlet-name>
        <servlet-class>
                com.yourdomain.projectname.server.MySQLConnection
        </servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>
                MySQLConnection
        </servlet-name>
        <url-pattern>
                /MySQLConnection
        </url-pattern>
</servlet-mapping>
```

The information in the `servlet-name` tags are fairly arbitrary, but must be identical in both, the `servlet` and `servlet-mapping` tags. Essentially, for every Servlet you have running within the application, you will need to set these two tags within the `web.xml` file accordingly. Under `servlet-class`, you will have to enter the fully qualified package name of your Servlet (in this case of the `MySQLConnection` file, as it implements your synchronous and asynchronous interfaces and extends class `RemoteServiceServlet` of GWT. Most importantly, you need to set the correct URL under `url-pattern`. The path there, is the same one that you have specified in your GWT XML configuration file. Remember your `LoginScreen` class? This one implements `EntryPoint` and establishes the MySQL Service Target. It also comes with a `LoginScreen.gwt.xml` file, in which you once set a `servlet` tag. In that tag, you specified the path of the package of the Servlet as well as a path. The path you have to set in `url-pattern` is the same one you set under `path` in the `servlet` tag in `LoginScreen.gwt.xml`. Sounds complicated, but it's easy... I promise.

4. Create WAR File.

I am strongly hoping that you are using Cypal Studio or some other GWT plug in for Eclipse ;-) Otherwise, you might have to go about creating this WAR file manually. This can be quite the challenge and trust me – you don't want to do that by hand. Thankfully, Cypal Studio can help there. First, clean your project by going to `Project` then `Clean...` in Eclipse's menu bar. Select your GWT app project and Eclipse will clean, compile, and build your project. Next, select your project in your Project View of your workspace. Then, in Eclipse's menu bar, click on `File`, then `Export...`. In this Dialog, go all the way down to where it says `Web` and click on the plus sign to open whatever is underneath. And this "whatever" should be an item called `WAR file`. Click on it and select `Next`. Follow the Wizard and safe the file to where ever you like. Congratulations, your WAR file is created! By the way – a WAR file is essentially just like a JAR file with a special folder structure inside. You can even open it with your favorite Zip compression tool or even `jarutility` that comes with Java.

5 Install MySQL into Tomcat.

This step is really simple. Locate the MySQL Connector/J Driver that you have added to your GWT project's class path in your file system If you can't find it, that's kind of bad. Refer to my first GWT tutorial for a clue where you might have put it or get a new version at [12]. This file needs to be both in your GWT Application's class path on the Tomcat server, as well as in Tomcat's common libraries folder. The first is done automatically, if you created the WAR file like I said in section 6.4. The latter is done very easily, by simply copying the jar (should be something like `mysql-connector-java-5.1.6-bin.jar`) into the `$TOMCAT_HOME/common/lib` folder of your Tomcat. `$TOMCAT_HOME` refers to the path in which you have installed Tomcat.

6. Install WAR file into Tomcat.

This one is just as easy as putting the MySQL Driver into Tomcat. Simply take the WAR file from where ever you told the Wizard in section 6.4 to put your WAR file and copy it into `$TOMCAT_HOME/webapps` folder of your Tomcat. That's it.

In fact deployment is complete at this point, but let's just keep working for just a little bit to verify that stuff worked.

7. Launch Tomcat.

This step is going to be simple, too. I could actually leave this one out, but I decided to keep it in here for completeness.

Once the WAR file is deployed, Tomcat will take care of the rest. This even works *while Tomcat it running*, which is called “hot deploying”. Just to be safe, we will shutdown the server and restart it.

To shutdown, simply execute the `shutdown.sh` (on Linux/Unix or Mac OS) or `shutdown.bat` (on Windows) scripts in the `$TOMCAT_HOME/bin` folder.

To start the server again, execute the `startup.sh` or `startup.bat` script in the same folder. Easy as pie.

Tomcat should now deploy that WAR file by unpacking everything therein into a directory with the same name as your WAR file. **Existing files will not be replaced!** So make sure that it is empty or make sure that you know what you are doing. If you want to replace a previous build, just delete the existing directory. If you want to test two different builds concurrently, just rename the existing folder or the new WAR file to something else. You get the idea, I am sure.

8. Verify success of Deployment.

Now that Tomcat is up and running again, your GWT Application is deployed, and hopefully Tomcat didn't complain during deployment, it is time to verify that it works. In your favorite browser, go to the URL of your server, with the port Tomcat is running on and to the folder, you deployed to. This may be according to this pattern:

`http://servername:port/warfilename/EntryPointFile.html`

`servername` here stands for the URL you can reach your server under. This may be `localhost` or something like `yourdomain.com`.

`port` is the port that you have set Tomcat to run on. You can manipulate that in Tomcat's `server.xml` file. Pardon me for not detailing this procedure, but this would be beyond the scope of this tutorial, as Tomcat configuration can be quite a pain. So much be said, though: typical port numbers would be something like 8080, which is the dynamic web server port default, or 8100 (which is the JBoss default – and JBoss is yet another Tomcat flavor). I believe the Jakarta/Catalina default (Catalina is the code name of version 5.5 release of Apache Tomcat) is 8010, so you may assume that. Consult your `server.xml` in the `$TOMCAT_HOME/config` folder for the exact port.

`warfilename` is straight forward, right? It is the name of the WAR file that you entered into the Wizard in section 6.4. Unless of course, you changed it during following the steps in section 6.7. Remember that when hot deploying, Tomcat will extract the contents of the WAR file to a folder with the same name.

`EntryPointFile.html` is a little bit more special. As you may recall, your GWT application has a class that implements the `EntryPoint` interface – you know, the one with the `onModuleLoad()` method. This class name depicts also the GWT XML configuration file. During build, the GWT compiler has created an HTML file by that name and this is precisely the one you want to call in your browser. If you followed the example from the other tutorial, this would be `LoginScreen.html`. Hence, also according to that tutorial, your URL would be

```
http://yourdomain.com:8010/projectname/LoginScreen.html
```

You may leave out the HTML file from the URL, if you like. Then, Tomcat will show you a file listing of all files in the folder `projectname`. There is a lot of stuff in there that may sound like gibberish to you, but yet again, you want to look for a HTML file named like your `EntryPoint` class.

Summary.

OK, that's it! If everything worked out, you should see your GWT application in your browser, similar to what you were able to see in the hosted mode using GWT's own internal IDE server.

Let's recapitulate for a minute. In this tutorial, you have seen how to modify your GWT application to tell the dynamic web server about the Servlet, how to deploy the necessary resources and the GWT application itself, and how to verify that it works. Essentially, we enabled Tomcat to be able to pull data from some MySQL server, if some deployed GWT application that specifies a MySQL connection Servlet requests so.

But wait a minute... didn't we forget something? How about telling Tomcat were to find the MySQL server? How does Tomcat know where to pull the data from? Well, that is actually part of your Servlet. If you do it right, it won't be necessary to configure Datasources or JDNI's anymore (whatever that is) – this will be added to the *"context" of the deployed GWT application during hot deployment*. Tomcat, ergo, does it by itself. You will specify this in your GWT project, when you write the Servlet. Details can be found in section 6.8 of the other tutorial.

Well, folks, that's it. I hope you like this little tutorial and find it helpful in your next GWT programming venture. Let me know of any successes, failures, and whatnot.

References.

[1] Dewsbury, R. (2007). *Google Web Toolkit Applications.* Boston, MA: Addison-Wesley Professional.

[2] Geary, D. with Gordon, R. (2008). *Google Web Toolkit Solutions.* Boston, MA: Prentice Hall PTR.

[3] Google's GWT MySQL Example Project, accessed October, 24th, 2008
http://code.google.com/p/gwt-examples/wiki/project_MySQLConn

[4] Google Search Engine, accessed October, 24th, 2008
http://www.google.com/

[5] GWT – Google Web Toolkit, accessed October, 24th, 2008
http://code.google.com/webtoolkit/

[6] GWT Download Section, accessed October, 24th, 2008
http://code.google.com/webtoolkit/download.html

[7] The Eclipse Project, accessed October, 24th, 2008
http://www.eclipse.org/

[8] Eclipse IDE v3.4 "Ganymede", accessed October, 24th, 2008
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip

[9] Cypal Studio, GWT Plug-in for Eclipse Ganymede, accessed October, 24th, 2008
http://www.cypal.in/studio

[10] Cypal Studio Documentation, accessed October, 24th, 2008

http://www.cypal.in/studiodocs

[11] WAMPServer – Apache, MySQL, PHP on Windows, accessed October, 24th, 2008

http://www.wampserver.com/en/

[12] MySQL Connector/J, accessed October, 24th, 2008

http://www.mysql.com/products/connector/j/

[13] GWT Serializable Types, accessed October, 24th, 2008

http://www.gwtapps.com/doc/html/com.google.gwt.doc.DeveloperGuide.RemoteProcedureCalls.SerializableTypes.html

[14] Niederst, J. (1999). *Web Design in a Nutshell, 1st Edition.* O'Reilly.

[15] Silberschatz, A., Korth, H. F., Sudarshan, S. (2006). *Database System Concepts, 5th Edition.* McGraw-Hill Higher Education.

[16] Apache Tomcat version 5.5, accessed November, 28th, 2008
http://tomcat.apache.org/download-55.cgi