Development of an educational role playing simulation: Case study of software development life cycle

Yina Arenas, Tamal Saha and Mona Sergi
{ypa5q, ts4rq, ms4bf}@virginia.edu
Computer Science Department
University of Virginia
Charlottesville, VA 22904

***Abstract*—Software engineering is a highly intellectual activity that depends on the creativity of the people involved in the process. As our understanding the software process model and methodology have matured over the last 50 years, we still have not developed a sufficient understanding of software engineering to develop large scale software systems. As graduate-level students of a software engineering research and practice course, it is very important to have some practical experience of software development life-cycle to better understand the material studied in the literature. Moreover, using publicly accessible source code management systems for both commercial/non-commercial projects and communication among team members usually via the Internet have emerged as a new paradigm for collaborative software development. To this end, we chose *Nanosim*, a role play simulation of the National Nanotechnology Initiative as our case study project. In this project, we have collaboratively completed the detail design and implementation of the participant's mode of *Nanosim* describing the complete life cycle of the software following an Spiral/evolutionary approach and a layered event driven architecturaly style using GWT [1].**

I. Introduction

NanoSim is a role-playing simulation game with a precise and carefully thought out educational purpose. These types of simulations have been used in social sciences, business, and military and political studies for a long time, but rarely do they engage on science and technology issues.

In this paper we describe the general concept of *Nanosim* and the approach we took to re-design a part of the system. We describe the motivation for the various decisions that we were faced to make during the development process as well as the risks and constrains that we had to overcome. We give special emphasis to the architecture of the system and the implementation as well as the tools we used and their advantages. Overall we cover how we used this project as a case study for the achievement and application of the software engineering practices we learned through the course.

II. *Nanosim* General Description

The risks and opportunities arising from emerging technological and business systems cannot be entirely foreseen. For purposes of such prospective analysis the participation of social scientists is required alongside the technical scientists as stakeholders at the forefront of innovation. Such an involvement is particularly critical during the earliest stages of development of technological systems such as nanotechnology, because the maximum degrees of freedom for changing technological direction occurs at, or just after the point of breakthrough; and that is also the point where the long-term implications are hardest to visualize. However, collaboration across these disciplines can be successful only if scientists, engineers, and ethicists can communicate meaningfully with each other.

Nanosim is a tool developed to train second-year engineering students at University of Virginia [2] on the development of "interactional expertise" as a method for such collaborative communication. Interactional expertise is the ability to adopt the language and concepts of an expertise community sufficiently to pass as a member, without being able to do the actual research [3, 4]. An expert within a community of practice knows which problems are worth pursuing and also how to solve them in the paradigmatic way. Unlike the domain experts, the interactional expert can also take an outsider's view of the community, looking at it from the standpoint of another paradigm.

In *Nanosim* participants are placed in groups representing the major decision stakeholders in nanoscience and nanotechnology such as: government funding and policy regulatory agencies, industrial and academic research laboratories, as well as non-governmental think-tanks. These groups start with some basic capabilities and work their way through a technology tree of "toolkits", "prototypes", "technologies" to eventually address "grand challenges" such as curing cancer, and human enhancement. Towards this purpose, research labs need to present technical proposals to government agencies to fund their ideas, the government agencies need to develop regulatory policies for the multifunctional technologies under development, the new ideas need to be patented and the non-governmental organizations need to study the risks. The ABET guidelines [5] emphasize teaching students to work in multidisciplinary teams. *Nanosim* reinforces that goal, but in addition, teaches students to work with other teams representing different roles and perspectives, forming trading zones and gaining enough interactional expertise to see how the simulation looks from the standpoint of a group with a

different mission.

*Nanosim*s project is build up in the junction of different disciplines, first the software engineering of the distributed web-based platform and database system, second the production of a realistic technology tree in nanoscience and nanotechnology. And last the educational component that involves production of techniques and metrics for the assessment of student's learning process derived from using the tool.

The structure of *Nanosim* is composed by three main parts, the first being the administrative mode used for maintenance and continuous development of the system. The second is the instructors' mode in which the instructors can setup the *Nanosim* scenario. And the third is the participant's view which depends on the category of the group he or she belongs to. In the participants mode the groups are broken into four general categories:

- Research (Labs)
- Government (Funding Agencies)
- Regulatory (Consulting Agencies)
- Newspaper

These four types of groups have different roles within the simulation but must interact in order to reach higher level system goals. In this paper we will cover the software component of *Nanosim* and describe the process taken during the course of 8 weeks where the participant mode of *Nanosim* was redesigned as part of the authors' project for our graduate Software Engineering class.

III. Background and Motivation

Nanosim's concept was conceived three years ago by Prof. Michael Gorman from the Science, Technology and Society department of University of Virginia. It was initially developed as a basic web service and database driven application. The user interface was implemented in PHP and was hosted on the University of Virginia servers. The initial version of the software presented major on-the-run crashes that lowered the performance of the tool in the classroom. It required the developer to be present while the simulation was running to fix the software on the fly. Moreover, simple modifications to various parameters and configuration of the simulation had to be hard coded, resulting in an impractical complication for the instructor.

We started the project considering that the redesigned software platform of *Nanosim* should be robust, adaptable, portable and user friendly. This is a project that will continue to be developed and used in a classroom setting at UVA. Therefore it is important to have a solid foundation that will allow the software to grow and be adopted by others.

Initially, we were faced with the decision of fixing the current code or starting from scratch. After a short consideration we decided to go for the second option as it would be more suitable for the organization and design of a solid architecture and it would also fit more for our own pedagogical interest within our Software Engineering course.

During the 8 weeks we spent for the project, we devoted significant amount of time to the design of the architecture of the system, taking advantage of technologies like Google Web Toolkit (GWT). In the following sections we will describe our work in more detail, starting with the selection of the process model and continuing with the risk analysis, requirements and specification, architectural view and design, implementation, the evaluation of our experience and conclusion.

IV. Process Model Selection

The objective of this section is to illustrate the process of selection of the process model for the development of *Nanosim* from the various software process models available. Initially *Nanosim* was developed in a code and fix fashion which evidently presented several difficulties for the end users and the instructor, as we mentioned before, there was a need for the developer to be fixing on real time the buggy software and the code was poorly structured. Our first consideration was to follow a waterfall/stagewise and sequential development model which emphasizes the completion of each stage and fully elaborated documentation before moving to the next. But considering the risk of not having a working system in the allotted time frame and the possible changes in the requirements of the system, we ruled out this methodology, mainly because it did not gave a systematical treatment for risk mitigation and we could have ended up with only documents but not a working evolutionary prototype.

Our main concern for the project was not the completion or fulfillment of the entire requirements set of *Nanosim*, but the development of a solid platform that could grow overtime and would be robust and portable. Based on this, we considered following spiral/evolutionary model [6] where we took early care of risk mitigation, planning for changes and analyzing the big decisions and early commitments to technologies that we were faced with. Although there is an additional overhead associated with this development model, the benefits are seen in a smaller development effort and earlier quality feedback, we tried to have tight loops and a working system with new features in a weekly basis.

V. Risk Analysis

Having chosen a Spiral like model, our first step in this quest was to analyze the risks involved in the project. Initially our main concern was the risk of underestimating the complexity of the project. The project by itself was well-defined, and as it already had a working prototype, the requirements and even specifications were well stated, this means the overall idea of what had to be done was clearly understood. However, we still faced several difficulties in re-developing the project. We list a summary of the risks we faced, and our mitigating actions, below:

1) Scope and Time management: the original definition of our project was too broad to fit in an 8 week time frame. The question was what part of the project was doable during this short period of time. To mitigate this we focused only on the participant's mode of *Nanosim* and again since we were more concerned about the architecture of the system, we developed the different layers to be as neatly and complete as possible so a new feature could be added easily.

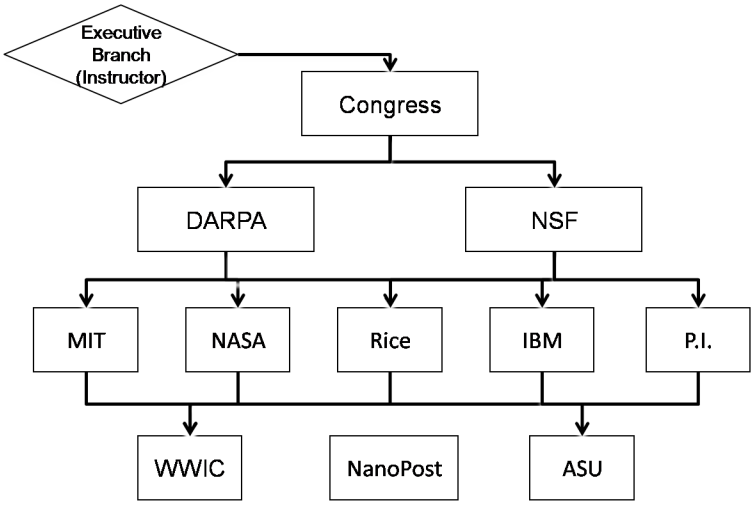

Fig. 1. NanoSim groups

2) Experience: Another big risk that we faced was the non-overlapping and different background of the team members. The project was educational, so we were not forced to use any specific language, database, or software development environment. This made it hard to come to a conclusion regarding which technology is the best fit for this particular project. We decided to keep the same database technology of the original system and focused on adopting a new tool for the business and presentation layers that would have a small learning curve (based in a technology already familiar to all the team members) and that would simplify some of the problems faced in developing web applications.
3) Github: Although we divided the tasks in a way that there were the least amount of conflicts among the files each of us was going to work on, we faced lots of difficulties in working with this service. At some point in our project, we were convinced that Github was more of a barrier for us rather than a help. So, we changed our code management service to subversion.
4) Flexibility: As we mentioned earlier, this project is going to be continued and new features will be added. Therefore it was essential for our project to be flexible and extensible. To ensure such property, we used Google Web Toolkit (GWT) and followed a layered architectural style (dao, server, model, and client). Moreover, as we divided the project to different parts, the different parts of the project became more modularized.
5) Correctness: As we discussed in class, testing is a challenge for any more-than-a-few-line project. Unfortunately, the time frame did not allow us to develop a complete test suite of the project, remaining as a risk that we were not able to address completely.

VI. Requirements and Specification

After completing the risk analysis part we continued with the refinement of the requirements and specifications of the part of *Nanosim* that we were focusing on. In this document we will briefly comment on the gist of our analysis.

Nanosim's general requirement is to be a hands-on simulation environment that offers:

1) For the developer: a robust platform on which additional features can be easily implemented in future.
2) For the instructor: simple and quick setup of a simulation
3) For the students: a user friendly environment with an eye to the fulfillment of the pedagogical goals.

The system allows access for participants in different geographical locations and presents content according to the participants placement in different groups that represent the different stakeholders in the simulation; each group has a certain type and a defined set of actions and properties.

A key point that we kept in mind were the non functional requirements of the system being extensibility, flexibility, interoperability, and maintainability of the code base.

The specifications of the system include database independency and the facilitation of different services to the end user like: an access control system to support the login of registered participants to their group's home page and the presentation of the appropriate content. An internal mail system that supports the participant's communications. A budget transaction system that allows money to be sent-to and received-from the different groups. A proposal submission system that allows the groups to submit, review and do research on the technologies. And a patent system that allows submission of patent requests.

Our next step was choosing an architecture that would satisfy the specification.

VII. Architecture

Based on the previous version of *Nanosim* and the problems it had, we concentrated big part of our efforts in the design of an architecture that would mitigate the risk of developing a system that would lie in the same pattern as before. Our main focus was on the identification and definition of level of abstractions around the functional requirements. We specified the objects of the system and their interactions with each other making use of the benefits of encapsulation of data

and the inheritance functionality, which results in shorter and more organized code, and reduced complexity. To organize the classes and their interaction better, we layered the architecture, where each layer shows one level of abstraction [7].

Among the advantages in layering the architecture we identified the importance of the interoperability between different types of databases, platforms, and settings. This property is essential especially in web-based applications in which diverse systems need to work together. Another key advantage is the compatibility between different devices and systems, in our layered architecture, only the bottom layer is interacting with the system, helping to avoid system dependency wherever possible and providing accessibility to the tool independent of the platform of the end-user.

By layering the architecture, adding extra features or modifying existing ones can be done in an easier manner. In many cases, only one or few layers are changed (but not all of them). We had to ensure about this property as this project is going to be continued by possibly other team of software developers. Complexity in code would degrade significantly the next teams' performance. And it might even lead to redesign the whole system, as it happened from the previous version of this application. One last advantage of the chosen architecture is the considerable simplification of the design and debugging processes. Breaking down the whole model into different layers that can only interact with the upper and lower layers also simplifies the testing stage. It also enables different software developers to work at different levels of abstraction. For this layered architecture we divided the whole system into three layers: presentation layer, business layer, and data layer. The presentation layer is the topmost layer in the application, the interface between the user and the application. It shows the information related to the user, and gives him/her the ability to browse the application and use it. The business layer, or middle layer, controls the functionality of the system in terms of the services the application offers to the users. The data layer consists of the database servers. All the information in the application is stored in this layer. Better organization of the data in this layer results in better scalability and performance.

Although the overall system follows a layered architecture, the presentation layer itself is organized in an event-based implicit invocation architectural style [8]. It helped us to capture the asynchronous user interaction with the system at its interface. The additional benefit of using this style is reflected in the reusability support of the different components. Also fulfilling the specification by providing a structural method for introducing additional commands or features to the system just by registering a service.

In addition for the underlying data access layer we chose a blackboard style [8]. The database represents the current state of the system and the collection of independent components in the data access layer operate on the data store.

Figure 2 shows the logical view [9] of our project in initial stage. As we mentioned earlier, we divided the architecture to three layers: presentation, business, and data layer. In Figure 2, we did not make the boundary between the layers very clear. Data layer is shown in the first two rows, in form of Person, Group, and etc. For sake of simplicity, we didn't include other details in data layer here. The third row from the top is the business layer in which we offer different services and data interaction with the user: sending fund to other groups, reading newspaper, and etc. Finally, the bottom layer is the user interface, and its interaction with business layer.

However, later in the project we complemented and modified the above logic view, by adding the web service layer to encapsulate the communication between the client and the server.

VIII. Implementation

In this section we discuss the implementation of our project and the decisions we made in terms of technologies for each layer of the architecture (Figure 3).

The package structure of the code reflects the architecture diagram of the system. The database layer is represented by the model package. In the package model (Figure 4), we have a class for each corresponding data table in the relational model. This acts as the basis for object-relational mapping. The data access layer is implemented by dao and util packages. In this layer, we have one data access object for each corresponding model. The data access layer communicated with the database layer using the standard JDBC API. The Web Service Layer is implemented by the combination of client.rpc and server packages. The client.rpc package implements the service interface for the presentation layer. Server package implements these service interfaces by invoking different data access objects. The user interface is basically divided into two portions. On the left is the various commands accessible to the user. For the currently active command on the left, user can see a detailed view on the right. The various commands available to the user are controlled according to the group access control policy predefined in the database. Mail, research, patent, profile and transfer are the different packages implementing corresponding user commands. Events, icons and internal packages are used for internal organization.

IX. Implementation of the UI

A requirement of this software was to develop a portable system that can be accessed without installing any additional software on the end user's machine. To meet these requirements we decided to develop the front-end using some java script based browser oriented technology. One of the other major concerns was the programmer productivity. This is very important because we had a hard deadline and the time to delivery date was quite small even for a class project like this one. Developing javascript based solutions are hard because of the inconsistent java script support of among popular browsers like Internet Explorer, Firefox, Safari, Chrome etc. Moreover, since javascript is a dynamic, weakly typed, prototype-based language, finding bugs in java script code is a time consuming and laborious task. So, we were looking for some technology where we can write the code in a strongly typed OO language which can be compiled into generate optimized javascript code. Among the available options, a few proven technology options were Google Web Toolkit (GWT) by Google and Script# by Microsoft. But as this was an academic project,

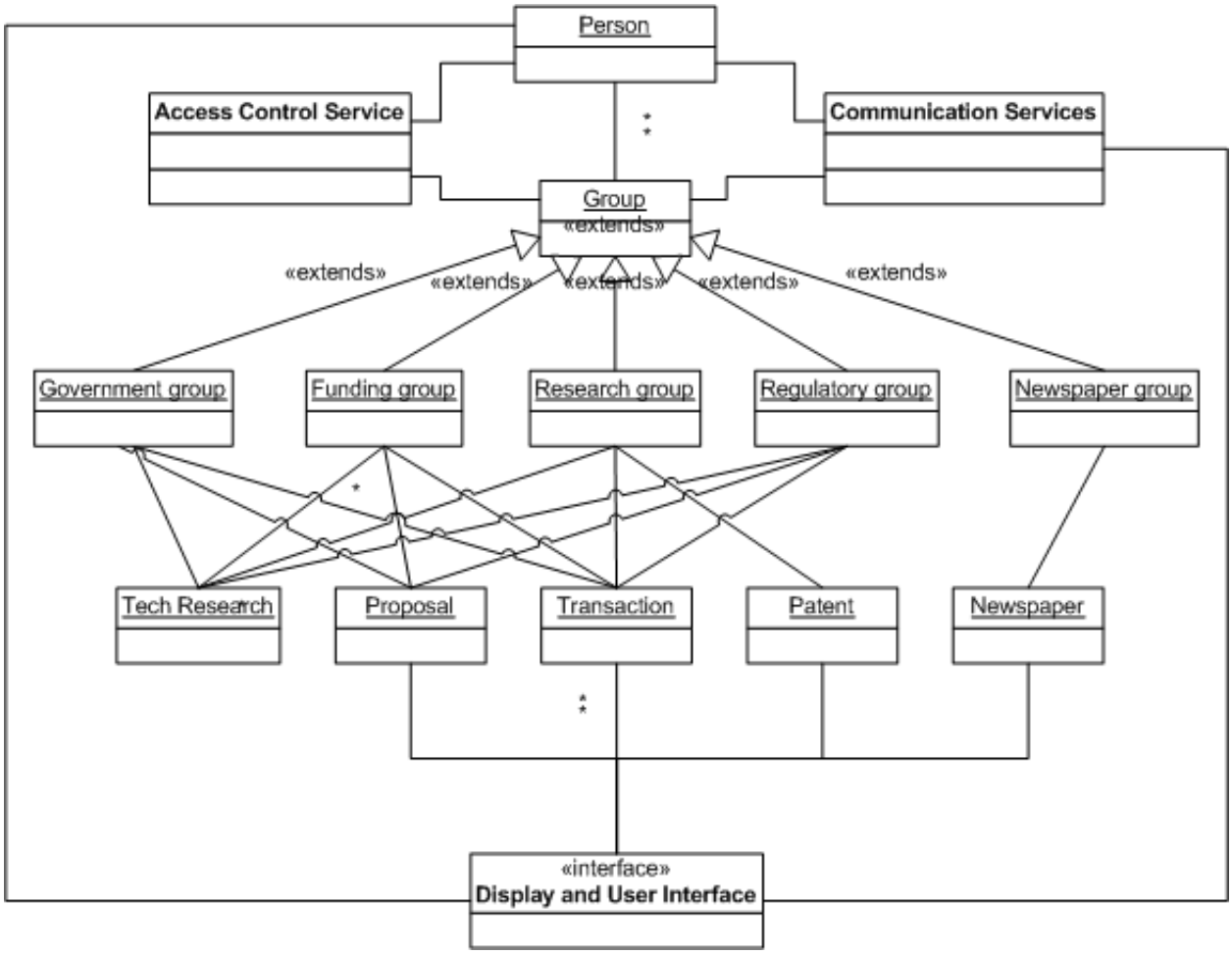

Fig. 2. Initial logical view

we preferred free and open source technology over proprietary technologies. So, we choose GWT as our front-end technology. Using GWT, we developed the source code in Java and compiled the code into javascript. Moreover, we used the popular Java IDE Eclipse and were able to take advantage of the debugging facility provided by GWT. GWT made it easy to develop reusable, efficient solutions to recurring Ajax challenges, namely asynchronous remote procedure calls, history management, bookmarking and cross-browser portability. One difficulty we faced with GWT is its lack of drag-n-drop user interface building feature. But its swing-like UI library with predictable behavior on the browser made it relatively easy over time.

X. Implementation of Back-end system/db

At the start of the project, we wanted to use ORM (Object-relational mapping) technique for accessing the back-end data store so that the actual physical database can be changed later to improve scalability, if necessary. So, we tested Hibernate as an ORM layer. ORM libraries work by adding some needed persistence information to the Model class, such as the associated session, the dirty state of the object and so on. But we hit the wall when we had to serialize the model class and send it over GWT RPC service. The problem was that serialization mechanism does not expect classes to have been instrumented or woven. So, when the response received from the client is de-serialized, the returned model class can't be mapped back to be original model class for auditing. To solve this problem, we tried an additional toolkit called Gilead. Gilead tries to address this problem by transforming persistent entities back to simple Plain Old Java Object (POJO), by removing code instrumentation and replacing persistent collections with their regular counterpart. But this library seemed to be unstable for production level use. So, after some trial, we decided to write our own simple wrapper interface for data-store access based on JDBC. As the back-end database technology, we choose to use MySQL, since it was already used by the previous version of *Nanosim*. It helped us to reuse the data already in there. It also met our goal to use free and open source technology.

XI. Implementation of Middle tire

In our project, middleware connects the client browser to the database server. As the middle tire technology, we had the option to use GWT RPC or J2EE. We decided to use GWT, since this is simple, easy to learn and comes by default with GWT toolkit. GWT-RPC provides the ability to send native and custom Java objects from the client-side Javascript code over to the Java server-side backend. GWT-RPC sends

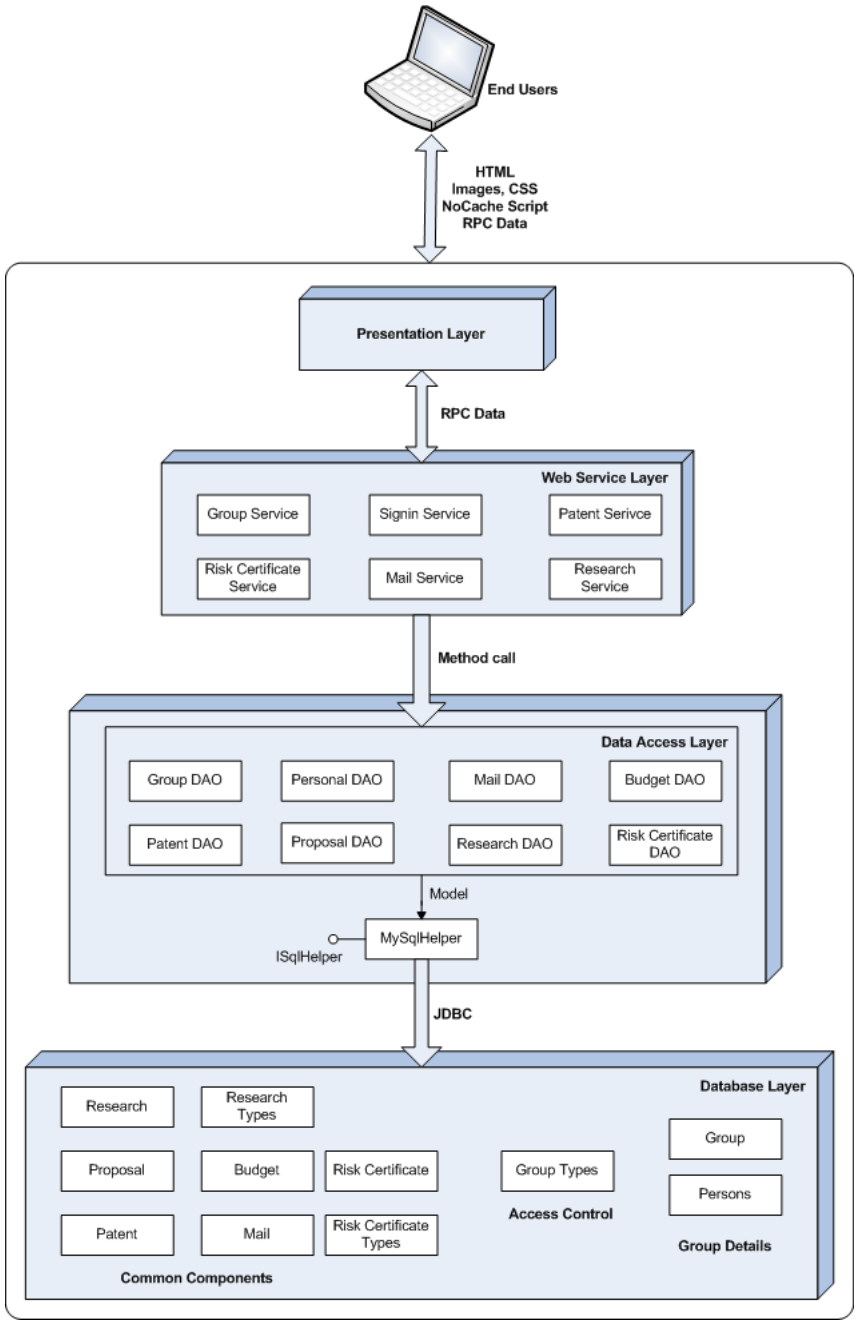

Fig. 3. Architectural diagram

a serialized stream containing the Class name, Method name, Parameters through the wire whenever an AJAX call is made. The calls are sent through the browser as a typical HTTP POST Request with the serialized method call in the body of the request.

XII. Source code Management

Although we mentioned about our experience with source code management in the risks section we consider it worthy enough to comment about the reason why we decided to switch back to svn for doing version control of our code.

Being a small team the use of Git did not bring any significant benefit to the development of our project; instead we did spent a considerable amount time trying to overcome the painful merges of our work. As the use of a decentralized SCM does not get rid of the need to coordinate updates in the master clone, it becomes very tedious if this updates are not done frequently.

After battling with Git we abandoned the decentralized approach and returned to using subversion. We took care of having practices that would not create a bottle neck for our development process (like always committing working code) which helped us avoid some of the common problems faced with centralized SCM.

At the end we came to the conclusion that even when using Git we were following a centralized approach and that it all

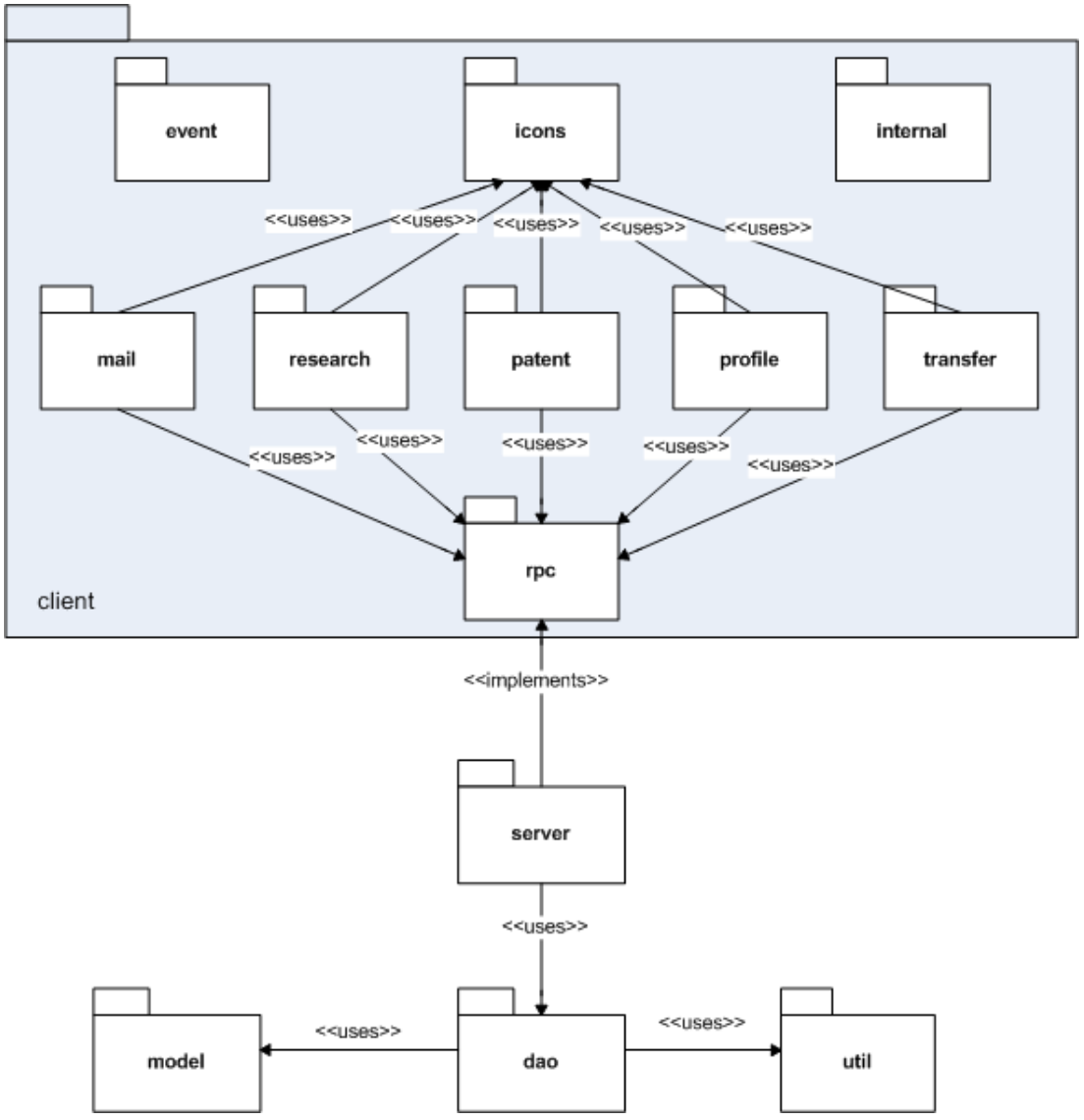

Fig. 4. Package diagram

came down to which tool was more "merging friendly", which surprisingly, for us was subversion.

The code is available in both github (git://github.com/yarenas/NanoSim.git) and google code (http://code.google.com/p/nanosim).

XIII. Testing

Due to time constraints, we were not able to extensively test the software. We have mostly performed functional testing during developing the code in an ongoing manner for purposes of performance verification and quality control. We have used the old system as a reference to verify the functions of this system. The next phase of the testing should be to do acceptance testing to review the functionality of the software against the requirements and specifications [10]. For future work also a user experience test should be practised to evaluate the usability of the interface.

XIV. Evaluation

In this project, we got the opportunity to develop an understanding of the challenges involved in developing software from the initial stage to the final release. It was a good opportunity to discover more about the papers we reviewed in class by doing an actual project implementation, and deploying the theory into practice. We also learned about new technologies including GWT, git, and svn. We got further insight on the interaction strategies and team work skills by using a collaborative development environment.

Overall, we believe this project resembled a real software project in industry in most parts. Our results show that we accomplished our initial goal in having basic functionality for the Nanosim software. We did our best to make the project extensible, and we expect an extended version of this software to be used in real world soon.

XV. Conclusions and Future Work

We have presented *Nanosim* and the process we took to develop a robust architecture of a part of the system that will allow future work to be continued and built upon the foundation that we have produced. We discussed the decisions and choices that we took as part of this process and the way we worked around overcoming constraints to successfully deliver a product that we believe contributes significantly to the overall *Nanosim* project. Furthermore we learned from all the different technologies we used and took advantage of the solutions they offer. We are aware that there is room for improvement and future work will include more documentation and extensive testing of the system.

References

[1] "Google web toolkit," http://code.google.com/webtoolkit/.

[2] M. E. Gorman, N. Swami, P. Warhane, J. Groves, and C. J., "Nue: Societal dimensions of nanotechnology: A course connecting communities," NSF NUE Grant Proposal, Tech. Rep., 2008.

[3] M. E. Gorman and J. Groves, *Nanotechnology: Societal Implications II, Individual Perspectives*. Dordrecht: Springer, 2007, ch. Training Students to be Interactional Experts, pp. 301–305.

[4] H. Collins and R. Evans, "The third wave of science studies," *Social Studies of Science*, vol. 32, no. 2, pp. 235–296, 2002.

[5] ABET, "Engineering chande a study of the impact of ece2000," http://www.abet.org/Linkedhange.pdf, 2006.

[6] B. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, May 1998.

[7] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, October 1992.

[8] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering, Volume I*. New Jersey: World Scientific Publishing Company, 1993.

[9] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, November 1995.

[10] Goodenough and Gerhart, "Toward a theory of test data selection," *IEEE Trans. in Software Engineering*, vol. 2, no. 2, pp. 156–173, June 1975.