Spring | 2015

Technical Report

Enclosed in this document is the technical report of the ***Open NEX project*** sponsored by ***Carnegie Mellon University*** and ***NASA***.

Academic Advisor:
Jia Zhang

Team:
Chih Hu
Juanchen Li
Gautam Madaan
Vinay Venkatesh

Table of Contents

1. Executive Summary

Collaboration is a requirement in most fields in the current world. Collaboration in terms of science could mean communication with others in the same field, sharing knowledge or sharing the research to help fellow scientists or academicians performing similar research.

NASA currently provides a platform called NEX for Earth Science scientists to collaborate, share knowledge and share research. This platform is not only intended to be a social network and project management tool, but also a tool where scientists can set up workflows and upload and share services. It is essential that all these components are cohesive. Currently, there are many security measures one needs to get through before making use of many of the features of NEX. To prevent this from being a hindrance to the scientists, NASA has another project called Open NEX which has a subset of the data from NEX, is hosted on Amazon Web Services and provides a subset of the services without this security barrier.

Open NEX will need to have data sent to it periodically if there are updates to the data that it contains. It will also need to serve as a social networking platform, a project management tool and a tool which incorporates other features like access control and workflow management and sharing of services.

Using Service Oriented Architecture (SOA), we were able to architect and implement two major components, the Social Network and Project Management., The Social Networking component provides the user with a network to connect with other users, demonstrate their work and conduct sessions within groups of users on various important topics. The Project Management component provides administrators and users the ability to create new projects, and assign tasks to other users and provides the ability to track projects through the lifecycle.

2. Motivation

The driving force and motivation for this project resulted from the wide variety of knowledge that we acquired during the course of our Software Architecture class, but which we had never applied in a real world project. Architecting this project, and applying the SOA-RA architecture reinforced our knowledge and provided us with insight as to the nuances with this architecture when implementing a scalable project.

Our other driving force was our passion to help NASA. We wanted to use our knowledge and help them in architecting this project, in implementing a POC and demonstrating the advantages of our design, which they then could incorporate into their design.

There were also new technologies that we wanted to learn such as the Play framework which helped us in becoming better developers and more knowledgeable in our field. Having to learn new technologies and implement a functional proof of concept with these new technologies was certainly a challenge that motivated us to work to our potentials.

3. Related work

There were three main sources that we analyzed in terms of related work for inspiration:

- Nasa Earth Exchange (NEX): Open NEX is essentially a subset of this already existing platform. The wide variety of documentation that this platform contains was certainly useful in understanding the kinds of data models that we would need to design for. This also helped us in understanding the subject matter. Even though we were not subject matter experts, analyzing this system helped us in identifying the goals for Open NEX.

- Jira: This is a project management tool that is widely used in the software engineering field. This provided us inspiration in designing the project management tool for Open NEX. Due to time constraints, we were able to design a subset of the project management features.

- Facebook: One of the largest social networking websites that is currently available provided us with the inspiration to design the social networking component. This helped us in identifying how to group our components and set up the structure for the different parts of the social network. It also provided us with ideas as to how to the network better.

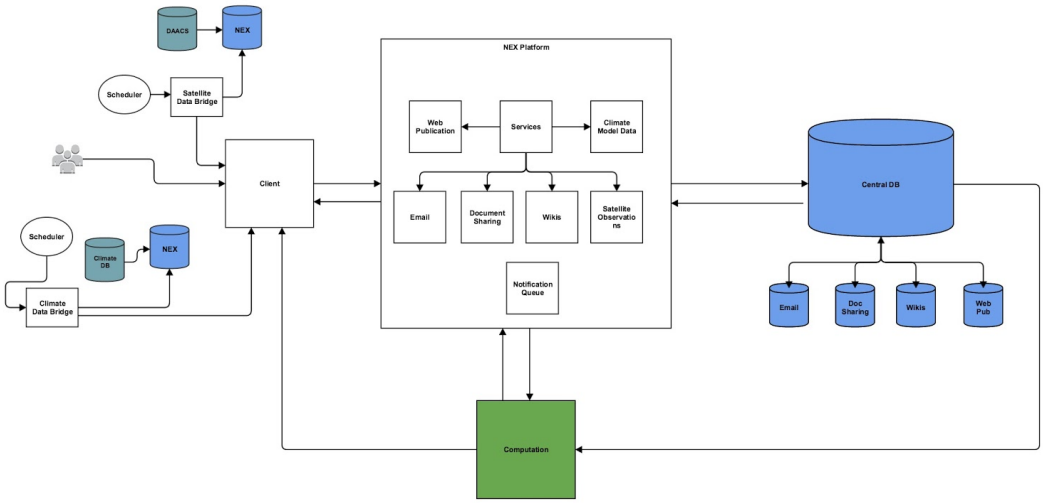
4. Assumptions & Considerations

Through out the design and implementation process of this project, we had several assumptions that greatly influenced our prototype. This includes:

1. Access Control : This was being implemented by a different team. So we had to mock a sample access control and assume that it would work in a similar way so that we could integrate smoothly.

2. The Relationship between our system components were analyzed and designed by our team. We did make some assumptions such as sessions only being associated to one group at a time. However, these relationships can be changed pretty easily using the Play models as needed.

5. System design



The initial design was to have services for each of the different components of Open NEX. There were data bridges that sent data from NEX to our databases through the client. All the services were on a platform, except for the computation service which interacted with the platform and was its own stand alone service. There were different databases for different purposes. Overtime we revised the granularity of various components and services until we reached a desired system architecture that supported Open NEX's collaboration platform along with its various requirements and took advantage of the nature of service oriented architecture.

Components

Our system consisted of five essential components. These components included Users, Groups, Sessions, Projects and Tasks. Users are the users of the system. Groups are interest groups that are created by users so that a particular topic or service that is used commonly amongst the users can be discussed and worked on within the group. Sessions are created by users or groups of users when an urgent matter needs to be discussed and the state of that discussion needs to be stored for the future. Projects are created for project teams that have an administrator that assigns tasks to other users and this can also be used in a corporate setting. Tasks are operations that need to be taken care by a particular user. Groups can consist of many users so there is a ManyToMany relationship between these components. Groups can have many sessions, but the general idea

was to have one group per session so that the session can be more focused. Groups can have many projects associated with it, but a project would mostly want to have one group associated with it. There can be many users in groups, sessions and projects. There can be many tasks associated with users, and multiple users can also be assigned to one task. This was done with the consideration that there could be people pair programming or working on the same task and in those situations, both the people would be assigned to one task.

For the sake of having a simple POC, we have simplified the relationships between the objects. The general object structure of the four components associated with a user are as follows:

Groups:

```
String groupName;
String groupDescription;
list<User> users;
User admin;
list<Session> session;
Project project;
```

Sessions:

```
String description;
Session session;
UserGroup userGroup;
list<User> user;
```

Projects:

```
String topic;
String description;
list<User> users;
String admin;
```

Tasks:

```
String description;
User user;
Project project;
```

View Points

To visualize the system design, we created different views of the system from various view points, namely the functional view, the information view, and the deployment view.

The Functional View

In the functional view, the NEX collaborator is the central piece of the architecture which interacts with all the users of the system and fosters social networking. It is also in turn responsible for authentication of users and the approval workflow. Project, group, sessions are all tightly coupled together and are also handled by the NEX collaborator as shown in Figure 1.1.

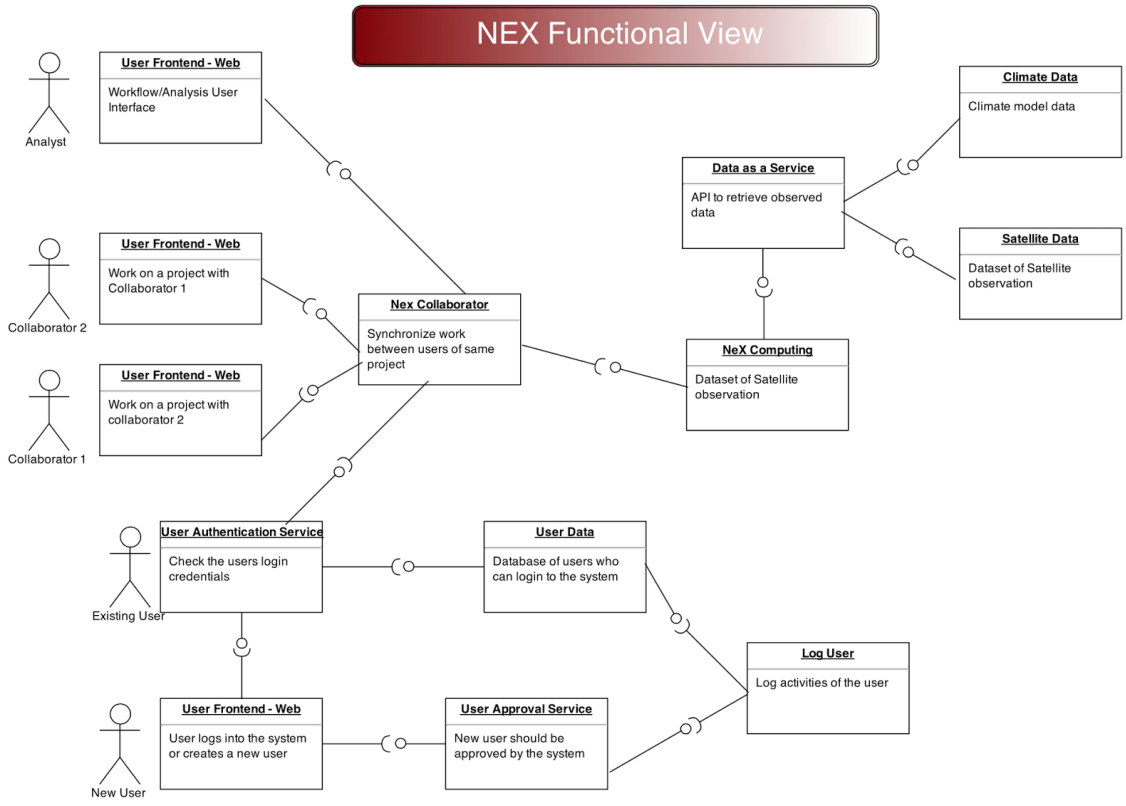


Figure 1.1

The Information View

The information view below in figure 1.2 represents the flow of data inside the system and the relationships among the different objects. The relationships between the objects are many to many. The below representation was refined continuously throughout the project and although the basic structure and relationships remained the same, the attributes of the states were modified as necessary.

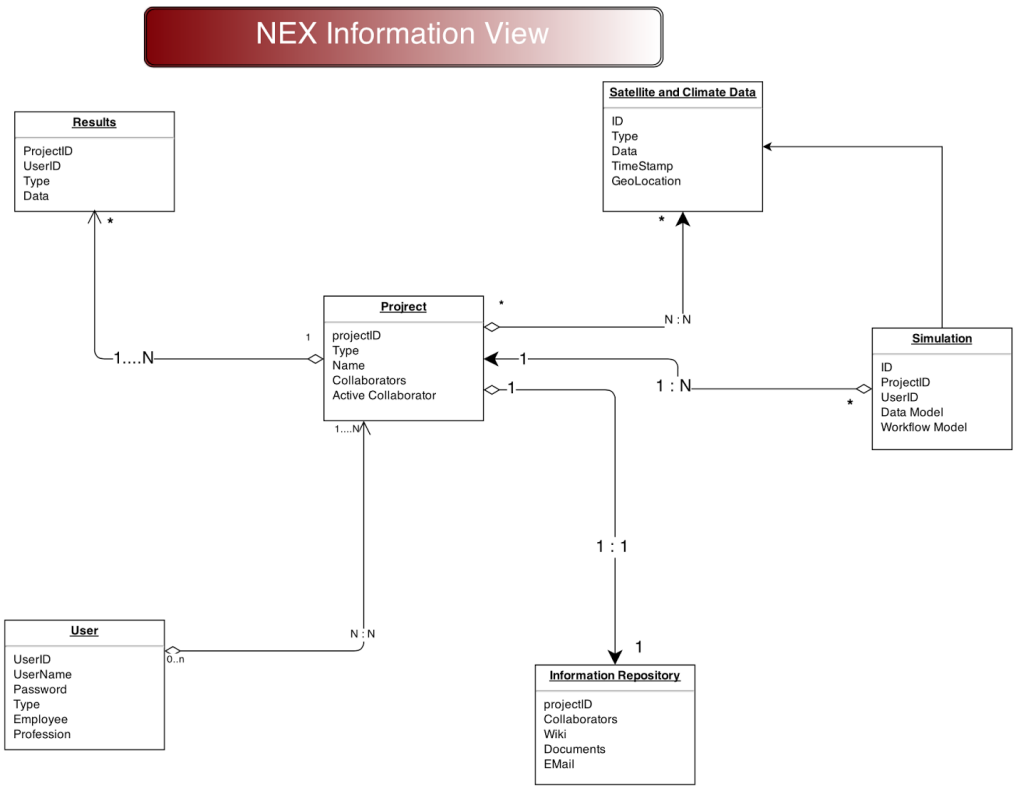


Figure 1.2

The Deployment View

After the above views were designed, we analyzed them to come up with a deployment view as shown below in figure 1.3. The constraints of the deployment were based off of a web service enabled architecture (Service Oriented Architecture). All the interactions with the backend servers are done via API calls through various services.

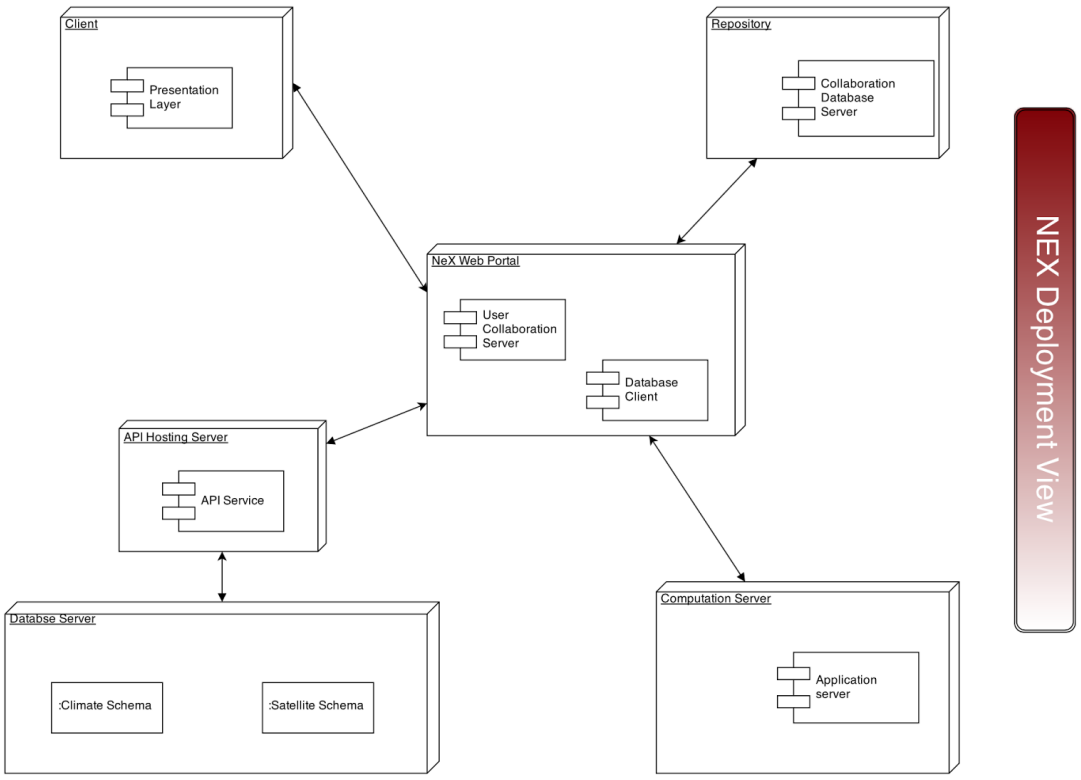


Figure 1.3

6. System implementation

Once all the views and high level implementation was done by the team, the next step was to implement the system and get a working prototype up and running. The constraint put on all the teams was the use of play framework which is an open source web application framework, written in Scala and Java, which follows the model–view–controller (MVC) architectural pattern**.** Our system components such as Project and Session have models associated with each of them which is the object representation of the actual component. The controllers contain the business logic, and the views display the information by deserializing the json and presenting the information in a easy to view way. Play is heavily inspired by Ruby on Rails and Django and is similar to this family of frameworks. Play uses Java to build web applications in an environment that may be less Java Enterprise Edition-centric. Play uses no Java EE constraints. This can make Play simpler to develop compared to other Java-centric platforms.

To enable interaction between the views and controllers, we used restful APIs. This is in conjunction with the SOA architecture. At the back end, we make use of an h2 in-memory database which is only available when the application is up and running. Although there are more configurations available with the play framework, for simplicity and prototype, we chose to use an in-memory database. Changing the database configuration is very easy with the framework as we just have to change a few lines in the configuration of the framework.

The user-interaction views are HTML pages populated using the scala templates. Scala works by passing data as objects from the JAVA application to the scala templates. All the object methods are available in the templates and can be accessed using simple getter methods. As is common today we used *git* for version control.

The Play framework makes use of JAVA annotations to create database models, define unique keys for database tables, and define relationships between database objects. The database CRUD operations are completely automated by the framework and are refreshed upon any changes to models. Moreover, searching the data in tables by unique keys are defined by a few simple lines of code which are standard for any application developed on the play framework.

The real challenge was similar to learning anything new. It took the team time to ramp-up and get the first user story up and running in system. After that it was relatively smooth. The main bottlenecks were troubleshooting the framework

configuration errors. In retrospective, we would have prefered to start developing the application or even a small prototype on play a little earlier during the development cycle.

For source code please refer to the Building & Running the Project section of the report.

7. Experiments and analysis

After implementing the social networking and project management aspects of OpenNEX as a prototype, we were able to receive some valuable insight into the architectural styles that it requires, and on how well the technologies we chose for the project matched up to the architecture and the system design. Our findings about the software architecture of OpexNEX fall into two major categories, one on the software stack containing the Play framework we used, and one on the RESTful architectural style that we applied to the system.

Play Framework

The Play Framework is an open source web application that utilizes Scala and Java. It follows the model-view-controller architectural pattern. This style of web framework differs greatly from the naive stack of websites which use simple HTML, CSS and Javascript. The back end models and controllers are backed with a Java or Scala backend, and the views are templated with a Scala based templating engine that outputs HTML dynamically.

Initially, this system proved challenging for our developers to learn. The Play Framework handles file routing and rendering in a much more complex fashion compared to the basic web stack, with a single route path hitting many files throughout the system. A part of this was due to the decoupled MVC architectural pattern that Play is based off of. Furthermore, the views are displayed through a Scala templating engine, which is a quirk to Play only, and our developers had to learn it from scratch as well. Another disadvantage that Play imposed on our development was the support. Because of the youth and complexity of Play, it was difficult to find suitable tutorials or examples built in the Play framework.

On the other hand, Play provides many advantages as well. The MVC design that it enforces is actually good for improving scalability within the design and the system. As the system gets more and more uses, it helps ensure that the system doesn't undergo an exponentially increasing amount of stress. Furthermore, the architectural pattern of MVC allows changes in the design to have less of an impact than changing data in a simple web design. Play also supports convention over configuration, hot code reloading, and display of errors in the browser. In addition, Play provides a variety of out of the box tools for web development. A particular feature that we leveraged in our development was the ability of Play to automatically create tables based on the models that we specify.

Overall the selection of using Play over a simple web stack was beneficial. While it was difficult to learn initially, the scalability and reliability it provides are quite beneficial. Especially as the project becomes larger in both the codebase and user base, the modularity will help accommodate growth. However, the lack of support is important to consider. Furthermore, it should be noted that there are many similar web frameworks in other languages as well, such as the popular Ruby on Rails, or Django for Python, and even Spring MVC for Java as well.

RESTful Architecture

Separating the backend and the front end with a RESTful service was another major aspect of our architecture. This design choice had a profound effect on our development process and on the system itself.

For our prototype, the RESTful architecture made development more complex. It adds a layer of complexity into our prototype. However, this means that we can easily split up tasks between developers within our team. Decoupling the backend and frontend allows members of the team to specialize in different tasks, and as long as there was a commonly defined API, the backend and frontend could be developed independently.

Furthermore, separating the frontend and backend allows us to back the system with a relational database. For our specific prototype, we chose to use the H2 database, which uses a form of SQL. Allowing our system to be backed with a database improves performance and provides additional scalability in the system when the user base grows.

8. Future Work

Beyond that, we would like to enhance the user interface by providing more extensive APIs to populate the groups, projects, and sessions appropriately. Then we hope to include a proper notification queue, such that each page the user is on, whether it's a user page, group page, project page, or session page, it will have a feed of relevant recent events. With this feature, we want to take advantage of the power of recommending and inspiring users to explore interesting topics and connect with other earth scientists.

One of the potential challenges of incorporating a social network into OpenNEX is accounting for the eagerness of earth scientists to create various interest groups and maintain them actively. To address this, we are proposing a workflow where the user is automatically prompted with a template group for creation initially, and then actively reminded to maintain the group's information periodically. The user will also be able to browse through existing interest groups quickly to see if a similar one has already been created. With this feature to enhance and encourage participation, we think the social network will be more active and users will find the collaboration platform to be more user friendly and supportive.

In addition, prior to releasing the prototype on the web, it is important that we focus on addressing the security concerns of the product. We started out with the basic assumption that all users of the site are friendly and here to collaborate, hence we are not strictly enforcing and checking user input fields. After the prototype is functional and well tested, we will then focus on adding security checks to strictly verify user inputs to ensure that we are not exposed to the various common security vulnerabilities such as authentication, clickjacking, cryptography, input validation, sql injection, and cross site scripting.

9. Conclusion

To support the OpenNEX project with a unified collaboration platform for earth scientists, we designed the system with service oriented architecture in mind, and implemented two of the six major components - Social Networking and Project Management. We utilized the play framework with MVC and supported RESTful APIs to account for integration with the other major components that are being implemented by different teams. This includes Access Control, Data Sharing, Software Sharing, Workflow Sharing, and Compute Power Managing. To support the the projects extensibility of new features through new components and services, we focused on the interoperability, usability, and modularity of each component.

With the current system architecture, design, implementation, and future extensions, we hope that the OpenNEX collaboration platform can encourage, facilitate, and ignite new ways for earth scientists around the world to conduct earth science research.

10. Building & Running the Project

Project Requirements:

1. Web browser
2. Play framework
3. Java Development Kit 6+

Steps to Build and Run the project:

1. Download the code from Github:
   https://github.com/cmusv-sc/OpenNEX-Team6
2. cd into the project directory
3. In the project directory start Play:
   >$ play run
4. Access the project interface with any web browser:
   http://localhost:9000

11. Contact Information

Point of contact: Vinay Venkatesh

Members information:

Chih Hu	chih.chi.hu@sv.cmu.edu	(510) 501-0353
Juanchen Li	juanchen.li@sv.cmu.edu	(808) 990-7710
Gautam Madaan	gautam.madaan@sv.cmu.edu	(408) 368-8895
Vinay Venkatesh	vinay.venkatesh@sv.cmu.edu	(408) 406-1324