Analysis Me

**Spring**

2015

Technical Report

Team 7

Enclosed in this document is the technical report of the OpenNEX sponsored by NASA.

# Table of Contents

1. **Introduction**
2. **Motivation**
3. **Related work**
4. **System design**
5. **System implementation**
6. **Experiments and analysis**
7. **Conclusions and future work**

# Introduction

President Obama announced several executive actions to reduce carbon pollution and help understand the earth science better. NASA Earth Exchange commonly known as NEX is a platform which will help scientist and student community understand earth science better. NASA has a collection of earth related data like climate, vegetation and pollution data. The idea is to make sure that all of the data collection effort does not go to waste. NASA also has a lot of computing resources which help in data crunching for data scientist working on a large dataset.

The goal of the NASA Earth Exchange (NEX) is to enable enhanced and more efficient use of Earth observations for NASA Earth science technology, research and applications programs. NASA Earth Exchange (NEX) is a new collaborative platform that brings together high performance computing cluster with large volumes (hundreds of terabytes) of NASA satellite and climate data as well as number of ecosystem and climate models. NEX facilitates end-to-end execution of Earth science research projects complete with data acquisition, analysis, model executions and result sharing. The open datasets are gathered across number of different projects. Then, the platform will provide a unified spatial-temporal schema that will give NEX users a better access to the large number of datasets from many NASA instruments and models. To enhance the user experience, NASA will inventory current NEX models, utilities and tools and package them together and provide it as a catalog of services to NEX researchers. Moreover, there will be a search and execute interface on top of the data and services catalog that will enable users to execute data queries, locate tools, models or libraries, provide them with information on integration with their own tools and finally execute their integrated codes on the NEX supercomputing environment. Because of the diversity of the NEX community, researchers often use different programming languages and development environments. In order to improve user interaction with the system, we propose to develop a set of client libraries that will enable users to access data and execute processing components directly from their environment such as MATLAB or IDL.For such a large scale platform there are a lot of decisions that need to be taken before we start developing the product. We also need to consider the risks involved and assess possible solutions beforehand. This significantly reduces the cost of maintenance if the risks are mitigated in the architecture itself. There are several aspects like scalability of the platform, security of datasets that need to be considered before implementing the components of a huge large scale system like NEX.

# Motivation

There are several factors that contribute towards a formation of a system like NEX. There are several players which have their own specific requirements. These are requirements then need to be finalized into the core functionality of the system. There are several other factors which affect the overall usage of the system which are not exactly an explicit part of the features of the system. The architectural trade off analysis methodology helps in converting real world requirements into software specific requirements on the basis of which the system can be built. We will discuss this in the following section and understand what motivates development of a huge collaborative platform like OpenNEX:

## Mission Drivers

The mission drivers are:

* Make the existing datasets available to normal users using internet
* Share the computing resources under NASA’s control with normal users
* Make the tools, algorithms and utilities required for data analysis available for normal users

## Business Drivers

The business drivers are:

* Performance
* Privacy
* Reliability
* Security
* Availability
* Usability
* Modifiability

Modifiability can be further divided into other drivers:

* Scalability
* Reusability
* Extensibility
* Interoperability(Should work on all client machines irrespective of different configurations, e.g. should work on Linux, windows and Apple Mac OS)
* Integrity
* Data Freshness

## Critical Functionality

The world now consists of large amounts of data. The most valuable thing in this century is also data. As one of the most important and hi-tech organizations in the world, NASA owns extremely valuable data.

NASA tries to share the valuable data to the whole world’s scientists. And also NEX provides NASA’s own data analysis models and tools.

NASA has this privilege to get the most up-to-date data, which normal scientists may not have the access to. On the other hand, even NASA owns the most powerful techniques, the whole world’s power and innovation is incompatible. So it’s a win-win strategy for the whole world.

## Constraints

NASA has its special security issue, so it’s hard for it to open up the whole database. First, NEX is not a total open community. Second, the access to data and computation facilities should be high secure level. For example, users cannot access unauthorized data, users cannot get super user control of the computation facilities.

## Business Goals and Context

NASA wants to share data with other scientists through NEX. So the primary goal of NEX is for developing more scientific value from the data to benefit the science. Anything that is commercial-intensive is not welcome here.

Still NEX will have billing units for charging the necessary facility maintenance. And it also is used to help NASA get more valuable data or create more helpful tools.

## Major Stakeholders

* NASA - The vendor who owns the HPC platform. As a service vendor, NASA wants to make NEX good for science develop and profitable. At the same time, NASA wants to follow the security constraints.
* Data Scientists - People who devise methods regarding simulation or data analysis. As the major users, scientists want easy access, because they may not be a computer guru. They also want most accurate and sorted data with powerful model and tools. They also want the connection to be good.
* Solution Architect - People who plan and design hardware/software components and their interactions within the system. As the core designer of NEX, architects should take into consideration of most important aspects, security, performance and scalable. Architects need to make the main point clear for the developer and tester to do their job.
* Software Developers - People who develop software and/or components which utilizes NEX services. NEX is big-data platform, so developers need to mix the cloud computing tech into it. And keep the security issue in mind when developing.
* Software Testers - People who make sure every software functionality works as expected. The most critical job for testers is to verify the security issue of the platform, especially data security and connection security.
* Public users - Other people who use the system, probably just to download the publicly available datasets or simulation results. They want to get the data and information with correct tags, so they can easily search and access what they want.

## Principal Quality Attribute Goals

The data can be shared quickly with all the world’s scientists. And the model, tools or data process should be efficient and scalable. And one of the most important aspects is to keep the whole system secure and functional.

## Architectural Approaches

The architecture approaches used in NASA Earth Exchange are listed below.

* Intel MPI and OpenMP framework - The system leverages highly parallel computation framework to speed up calculation process. This will save a significant amount of time to process each job for parts of the algorithm that can be parallelized.
* Virtualization - Provides “sandboxed” environment for every user and every session, ensuring isolation and security.
* Distributed processing - To be able to scale the computation power up and down depending on the workload, the system will process jobs in several distributed nodes. Moreover, this kind of processing also serves as a redundancy layer where a failing node can be transparently replaced by a healthy node.
* Layered strategy - This provides separation between software components when viewed top-down. Makes sure changes done in one layer does not affect the other layer.
* Load balanced - This ensures the workload are distributed evenly to the available worker nodes according to the load balancing algorithm. This will also ensure maximum utilization of each cluster nodes.
* Extensible system - Provides interfaces for 3rd party developers to add/complement existing features. Also ensures future system expansion can be done with minimal downtime.
* Shared unified storage - This promotes collaboration between NEX users, where every user can develop and store their simulation models for sharing with other users.
* REST based - The system can be utilized by its user through the provided REST API. The REST API exposes all core functionality to the end user with a simple request/response pattern.

## Utility Tree

The utility tree helps in identifying the key quality attributes and various scenarios in which they will have an impact on the system.

|  |  |
| --- | --- |
| **Phase 1: Quality Attribute Utility Tree** | |
| **Quality Attribute** | **Performance** |
| Attribute Concerns | The compute cluster should be able to efficiently process complex simulation models. |
| Scenarios | 1.        Given a large dataset, the user should be able to create a job that loads the dataset, apply a simulation model, and then process the dataset accordingly using the compute cluster service.  2.        The system should be able to queue all jobs and dispatch them to the cluster of servers according to a predefined job scheduling algorithm.  3.        The system should be able to leverage parallelism techniques (Intel MPI or OpenMP framework) to speed up any resource-intensive process. |
| Attribute Concerns | The compute cluster should be able to handle large traffic of data between servers. |
| Scenarios | 1.        Because of the sheer volume of the datasets, the network connection between the servers should have enough bandwidth to avoid bottlenecks. |
| Attribute Concerns | Performance is independent of number of simultaneous users |
| Scenarios | 1.        Irrespective of the number of users working on NEX at any point of time the performance of the system should not degrade to unacceptable levels. |
| Attribute Concerns | The amount of data varies. The processing should be scalable and suitable for big data. |
| Scenarios | 1.        When users want to process large amount of data, like annual weather, they can create scalable instance group to handle processing with just enough number of instances. |
| **Quality Attribute** | **Scalability** |
| Attribute Concerns | The system should scale well with the increasing amount of data stored/generated. |
| Scenarios | 1.        The datasets are gathered in a timely manner and will continue to increase over time. The storage system should be able to store all of the data.  2.        Every simulation will generate a large amount of data that must be properly stored. |
| Attribute Concerns | The system should be able to scale computing power up/down according to the amount of job/traffic. |
| Scenarios | 1.        Additional computing resource can be included to speed up parallel computing of data intensive calculations.  2.        Unnecessary computing resource can be detached to conserve energy consumption during idle period. |
| **Quality Attribute** | **Reusability** |
| Attribute Concerns | Datasets, simulation model, calculation results, visualization, and algorithms should be reusable by all users. |
| Scenarios | 1.        Datasets must be stored in a shared storage system for users to use.  2.        Simulation model can be shared among users.  3.        Previous calculation result must be available for reuse or further analysis.  4.        Visualization and renderings can be reproduced for future use.  5.        Algorithm can be shared and/or edited by other users. |
| Attribute Concerns | Any research can be reused and modified from its existing form maintaining all of its previous usable forms. |
| Scenarios | 1.        In case of refining existing research done on the NEX computing facilities, a version control system should be in place to maintain all the usable research outputs. |
| Attribute Concerns | The difference parts of the application can be reused (by other agencies for their own purposes). |
| Scenarios | 1.        Every tier is decoupled (portal-sandboxing-HPC)  2.        Portal solution is generic in nature and can be reused by other gov. agencies for other disciplines. |
| **Quality Attribute** | **Extensibility** |
| Attribute Concerns | The platform must support extensions to add more capabilities and open to future expansion. |
| Scenarios | 1.        The user can use their tools such as MATLAB or IDL along with the provided set of client libraries to leverage the HPC services.  2.        The system should be able to integrate with new component without impacting the availability of the service.  3.        The system must support addition of new HPC resources (like a new supercomputer) without interruption and without changing the API.  4.        System is ready to support multisite deployment and network infrastructure is adequate (ready to integrate an additional site other than AMES)  5.        System should support new technologies and changes in hardware and software. |
| **Quality Attribute** | **Interoperability** |
| Attribute Concerns | The system should support different types of data. |
| Scenarios | 1.        Multiple data sources may have different data formats and the system should be able to interpret the content.  2.        A data preprocessor can be used to convert the input data from one format to another. |
| Attribute Concerns | The platform should be compatible with the required software. |
| Scenarios | 1.        System software must be updateable without breaking current system.  2.        System software may be upgraded without affecting the availability of the service (minimal system downtime). |
| Attribute Concerns | Any subsystem can be integrated with 3rd party systems. |
| Scenarios | 1.        Third party databases can query data via a common API and receive response in a web friendly format.  2.        Third party data or compute resources can be integrated as an extension to the current resources, and can sync with the rest of the system. |
| Attribute concerns | NEX usage should be independent of client machines. |
| Scenarios | 1.        Any user on Linux or Windows or Mac OS should have no problems using NEX. The experience of using NEX should be uniform across OS platforms |
| **Quality Attribute** | **Privacy** |
| Attribute Concerns | The system should properly store sensitive data or share the data according to the ACL |
| Scenarios | 1.        The user may have private data in their home directory that should not be viewable by other users.  2.        The user may opt to share his simulation result for viewing by the other users. |
| Attribute Concerns | Sensitive datasets must be modified in order to protect privacy. |
| Scenarios | 1.        Sensitive data must be anonymized if required and if possible  2.        Noise is added to sensitive data for obfuscation if required and if possible. (Using differential privacy methods)  3.        The required level of anonymization or noise must be agreed, indicated and applied by the system authorities. |
| **Quality Attribute** | **Reliability** |
| Attribute Concerns | Data is not lost |
| Scenarios | 1.        In case of a database node failure, data is still accessible through redundant nodes.  2.        In case of failure of connection between a data source and the database, the non-transmitted data is stored by an intermediary data hub and forwarded once the database is available again.  3.        Sample datasets are consistent with the actual data set. |
| Attribute Concerns | HPC is not disturbed |
| Scenarios | 1.        When there are no exceptions, the HPC resources allocated to a job continues to run uninterrupted for the reserved time. (No termination of compute nodes in the middle of computation like in AWS EC2 spot instances)  2.        In case of a supercomputer failure, an automated job scheduler can reschedule the remaining computation to other computing resources in less than 5 minutes. |
| Attribute Concerns | Data is uniform |
| Scenarios | 1.        When data sources send invalid data, invalid data is discarded |
| Attribute Concerns | Jobs should continue to run in case of hardware/software failure. |
| Scenarios | 1.        Job execution should be transferrable to a healthy node(s) in case of failure on hardware/software. |
| Attribute Concerns | The system should have a backup power supply in case of power outage. |
| Scenarios | 1.        In case of power outage or brown-out, user activities must not be affected and every job should continue to run without any interruptions. |
| Attribute Concerns | Auto-healing functionalities are enabled |
| Scenarios | 1.        For trivial software errors, the software platform is supposed to auto-fix the issues so that there are less manual work for maintenance. |
| Attribute concerns | With a given set of inputs the output obtained should be exactly the same every single time. |
| Scenarios | 1.        Any research done on the datasets leads to some output. The process involved in getting that output should be repeatable with the same results every single time. |
| Attribute concerns | Any fixes or changes to NEX should not results in any downtime |
| Scenarios | 1.        Any fixes or changes to NEX should be applied without any system downtime and should not hinder progress of the research done by different users. Ability to rollback to previous working version should be implemented in the system in case of any failures. |
| **Quality Attribute** | **Integrity** |
| Attribute Concerns | Any data must be protected from accidental/malicious tampering to ensure its integrity. |
| Scenarios | 1.        Every data must be accompanied with hash or checksum.  2.        Erroneous data transfer over network must be correctable by recovery bit. If correction attempts are failed then retry the data transfer.  3.        TCP must be used for transferring important data between applications. |
| **Quality Attribute** | **Security** |
| Attribute Concerns | Every application on the system should be given controlled/managed access to the other parts of the system. |
| Scenarios | 1.        The system should provide a sandboxed environment where interaction between applications can be regulated.  2.        Data traffic inside the system or inbound/outbound should be regulated via managed firewall. |
| Attribute Concerns | Sensitive information should be transferred via secure channel. |
| Scenarios | 1.        Every REST API call should be performed over HTTPS to ensure its security.  2.        Every terminal session should be secured using TLS.  3.        Every connection to data repository should be encrypted to prevent eavesdropping. |
| Attribute Concerns | Users must be securely authenticated to access the system. |
| Scenarios | 1.        Every user action should be checked for the necessary credentials against the ACL.  2.        Every user action should be logged to audit trail for future auditing.  3.        Any write operation to the database must be authenticated and re-authenticated periodically over a secure protocol.  4.        User authentication works 99.99% of the time. |
| Attribute Concerns | HPC resources must only be used in the purpose of related scientific projects |
| Scenarios | 1.        An act of using the HPC resources for commercial or non-related content (like bitcoin mining) must be detected by an automated monitoring mechanism.  2.        A user job running on HPC must not make network requests |
| Attribute Concerns | Users’ operation should not affect other parts of NASA system. |
| Scenarios | 1.        When users access the database or the computation resource, there should be no way the users will touch the real NASA internal resource. |
| Attribute Concerns | Malicious users can be detected and avoided. |
| Scenarios | 1.        When users are hacked, system can recognized the unfamiliar behavior and avoid their future access.  2.        When user wants to log in, they are asked to provide more information than username and password. |
| **Quality Attribute** | **Data Freshness** |
| Attribute Concerns | Every node in the cluster should get the latest revision of data in near real-time. |
| Scenarios | 1.        Two jobs accessing the same data should get a consistent read. |
| Attribute Concerns | The system must be able to maintain data versioning. |
| Scenarios | 1.        A user can choose which version of data he wish to use.  2.        A user can revert back data to an earlier version. |
| **Quality Attribute** | **Availability** |
| Attribute Concerns | Resources are accessible all the time except scheduled maintenance. |
| Scenarios | 1.        Web portal is accessible 99.99% of the time.  2.        HPC resources are up 99.99% of the time.  3.        Portal response time must be less than 3 secs. |
| Attribute Concerns | NEX does not become unavailable |
| Scenarios | 1.        With too many users using the NEX resources at the same time the system becomes unavailable when it cannot handle the load  2.        In case of a physical hardware failure, the system does not become unavailable for users |
| **Quality Attribute** | **Usability** |
| Attribute Concerns | Scientist users are not computer guru. |
| Scenarios | 1.        Users’ configuration should be straightforward and easy to setup. |

## Scenario Generation and Prioritization

During the design and implementation phase this will helps us in deciding the priority of several features of the system based on the how critical these features are towards satisfying mission and business drivers.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scenario** | **Priority (1 – high, 5 – low)** |
| **1** | A new set of NASA data can be imported to the NEX platform when NASA finishes the security procedure. | 1 |
| **2** | A scientist can register as a certain area expert and acquire some portion of data from database. | 2 |
| **3** | NEX can examine the request from scientist and grant them certain level of access. | 2 |
| **4** | Scientists can arrange and process their data via easy and safe way of interaction, such as web console. | 2 |
| **5** | Scientists can invoice scalable instance group to process the data. | 1 |
| **6** | The data has different tools or models to help process, especially when data amount is huge. | 1 |
| **7** | NEX can charge scientists based on their data usage or computation power usage. | 3 |
| **8** | Scientists can arrange their tasks via task queue, such as add task, change task positions, delete task. | 1 |
| **9** | Scientists can get offline messages such as email notification about the current status of the task or the completion of the task. | 4 |
| **10** | NEX has several duplicate server nodes for different network zones. | 3 |
| **11** | With too many users using the NEX resources at the same time the system becomes unavailable when it cannot handle the load | 1 |
| **12** | In case of a physical hardware failure, the system does not become unavailable for users | 1 |
| **13** | NEX resources should dynamically allocated based on number of online users. | 2 |
| **14** | NEX (data, hosting, computing) resources should elastically adjust to the demand whenever possible. | 2 |
| **16** | Any user without appropriate credentials should not be able to edit or work with the datasets. | 1 |
| **17** | Any research done by a particular member of a community can be shared easily with the owner’s permission | 3 |
| **18** | Any research can be reused and modified from its existing form maintaining all of its previous usable forms | 2 |
| **19** | With a given set of inputs the output obtained should be exactly the same every single time. | 1 |
| **20** | NEX usage experience should be independent of client machines/platforms. | 2 |
| **21** | In case of a database node failure, data is still accessible through redundant nodes. | 1 |
| **22** | In case of failure of connection between a data source and the database, the non-transmitted data is stored by an intermediary data hub and forwarded once the database is available again. | 3 |
| **23** | Sample datasets are consistent with the actual data set. | 1 |
| **24** | An act of using the HPC resources for commercial or non-related content (like bitcoin mining) must be detected by an automated monitoring mechanism. | 1 |
| **25** | Web portal is accessible 99.99% of the time. | 1 |
| **26** | The system must support addition of new HPC resources (like a new supercomputer) without interruption and without changing the API. | 3 |
| **27** | System is ready to support multi site deployment and network infrastructure is adequate (ready to integrate an additional site other than AMES) | 5 |
| **28** | Third party databases can query data via a common API and receive response in a web friendly format. | 4 |
| **29** | System should support new technologies and changes in hardware and software. | 2 |
| **30** | Portal solution is generic in nature and can be reused by other gov. agencies for other disciplines. | 5 |
| **31** | Sensitive data must be anonymized if required and if possible (using t-closeness privacy models) | 5 |
| **32** | Noise is added to sensitive data for obfuscation if required and if possible. (Using differential privacy methods) | 4 |
| **33** | The required level of anonymization or noise to be applied in sensitive date must be agreed, indicated and applied by the system authorities. | 4 |

# Related Work

NASA has 40 years of data, all of these big datasets are valuable resources for scientists to study the earth and environment. Right now, NASA is planning to share these datasets to the public to call for the entire community to examine the data.

As the first step of opening access to the public science world, NASA will release 1TB of weather data and put it on Amazon AWS S3 storage. But the data will not be the only interesting part of OpenNEX. In order to leverage this data, our three teams focus on six different modules, which are all around this 1TB data. These six modules share the same purpose, to make more value from this data, to create more interest for gathering people around this data.

## Share the service

Over the last 40 years, NASA scientists have developed many software/algorithms/models to analyze the datasets. Such resources should be shared in the community, so that they do not have to be recreated. And also how to let new user/scientist submit their own work and to be utilized by others, is also an important part for setting up the cooperation in OpenNEX.

First of all, all the services should have some mandatory attributes to describe the service so that others can clearly understand the basic information of this service, and also it’s helpful for the system to recognize and classify the service for better usage. Those attributes may include description of the service, type of the service, version of the service, license that the service may apply, credits for the contributors, tags for better identifications, and so on.

Second, these services should be independently hold by the users who submit them. The users will submit services as web services, which can be accessed via Internet, so anywhere in the world, people can access that. And it’s also easy to use through web API. OpenNEX is a platform for gathering these services and generates more value from them, but not to hold them.

And it’s better to have a unified format for the input and output, which will make the corporation easier, for example, combining different services into a workflow.

## Share the workflow

Scientists’ work always is based on other researches. So does this happen in OpenNEX. We’ll encourage the scientists to work with others via different means, one of which is sharing the workflow.

What is a workflow. Workflow is a flow consisting of many services from the OpenNEX platform. A workflow connects different services to make a bigger work. For example, scientists might need other service for data extraction before the real data analysis, scientist might also need other service to generate the result.

In OpenNEX, user can create a workflow via drag-and-drop from the existing services list, and connects them together. User can share what they create to the others. So workflow also needs some necessary attributes to describe itself.

Workflow doesn’t really exist on any platform, it is truly a script of the order of running services. So OpenNEX can help user to “execute” these workflows, which indeed is to call all the services with corresponding configurations, and pairs the input/output.

## Manage computing power

Moving code to the data is the trend of cloud computing. NASA’s data amount is very huge, and in the future, it will probably release more data to the public. So cloud computing integration for OpenNEX is inevitable.

Since the data is all on Amazon AWS right now, researchers can use AWS as a platform to process data, using some software (shared) on the dataset (shared). How to optimize the AWS resource allocation for specific workflows in OpenNEX is a problem We want OpenNEX to be portal for scientists to work with data, and scientists may not be an expert of cloud computing.

In this case, OpenNEX should act as a wrapper platform to simplify the management of Amazon AWS resources. For example, AWS can help user create a virtual machine, but AWS cannot set up the corresponding services for user. OpenNEX should utilize AWS’s functionalities, and leverages them into some high level features, like executing workflow, and auto scaling for different amount of data. And also OpenNEX can create better solution for the pricing, then user will be able to see all kinds of available VMs together with their pricing, in some categories. When a task (job, workflow) comes in, OpenNEX can recommend the combination of VMs for the user.

OpenNEX can also provide a centralized dashboard for scientists to have a detailed look at their workflows progress.

## Project management

Project management means to organize users from the social network into different projects. So that users can work together in the context of a project group. In OpenNEX, we emphasize corporation, we know that the scientific world needs group work.

Users of OpenNEX can create groups based on projects, and invites others to get into the groups. The group will give access to shared data, files and discussions for the project. Anyone in the group has the abilities to modify these stuff.

In this way, scientific corporation doesn’t have to be constrained by the location any more. Since the data is on the fly, the code is on the fly, work should be on the fly as the same. Expand the scope of corporation will definitely increase the chance of better ideas, especially for those weather and earth science topics, which are all global topics.

## Social Networking

Scientists usually don’t communicate that much. The most popular ways for them are either by reading latest papers or attending academic meetings, both of which are not that up-to-date. What if we create a online scientific community to include scientists in.

In OpenNEX, we want to helps users connect to groups and projects, and also know the progress of the others’ work, and the hot topic or trend for interesting dataset. Social networking will give all the scientists a new way of researching, which can inspire more people to get in and to appreciate the hard-work of scientists.

Also social networking in OpenNEX will serve as a gateway to the rest of NEX’s services, like a social dashboard which includes all the workflows, services, data, and project groups. We also want to include external social network integration into the system. Together with recommendation engines, social network will be intelligent from user perspective. In this way, first we can have a wider and bigger influence on the whole society, and second to interest more people into science, for example, children and students.

## Access control

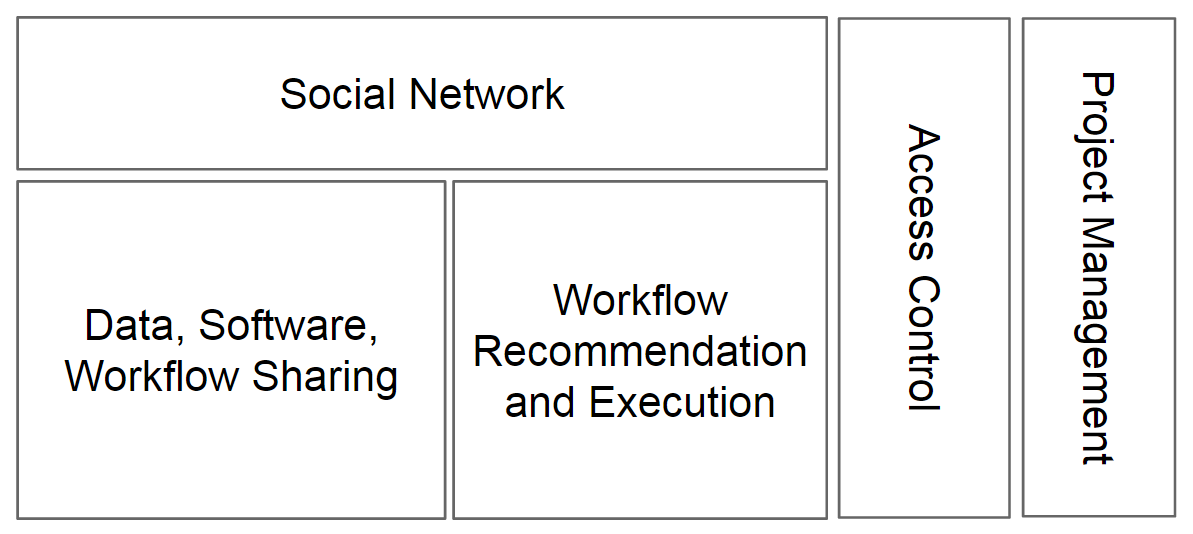
Unlike the other features, access control is about Non-Functional Properties. OpenNEX will use NASA’s resource and data, so security and privacy are among the most important properties. First of all, we need to limit the usage of the NASA resource to only certified users for scientific research. Secondly, access control needs to prevent user’s scientific result being stolen or misused by others.

Availability is also very important for people to access NASA data. It is always frustrating for users to get denied when they want to access the data. If it’s not always available, then it loses the meaning of putting data on the fly. Concurrency is important because the system should allow many people to access simultaneously.

## Overall design

The system will have several layers, the social network layer to be the portal of all other services and the main UI. Under that, data, software, workflow engines will provide the corresponding data, software and workflow services. Together with the recommendation and search engines, we can generate more values from the existing services.

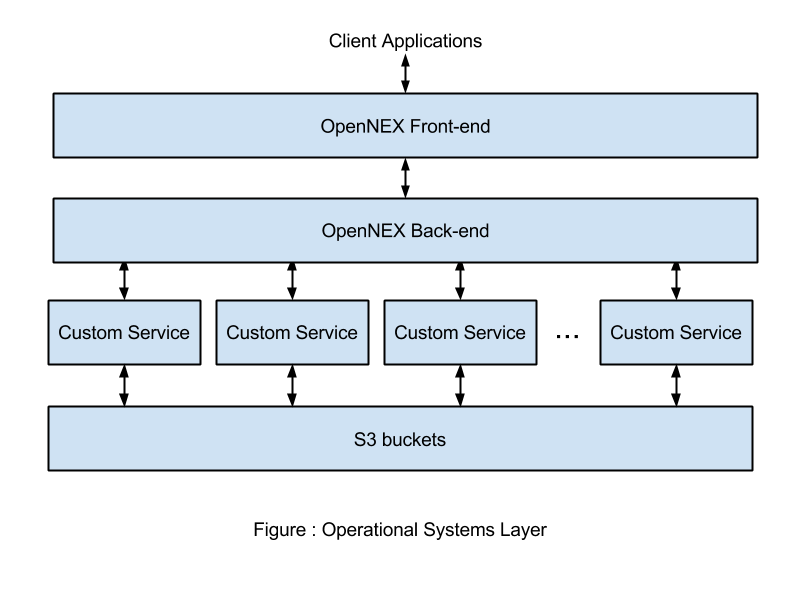
In the vertical side, we need the access control layer to control the whole system for security, privacy, availability and concurrency. Beyond that, we have a project management to use access control for more use, like project access, group manage.



# System Design

The overall “sharing workflow and software” sub system follows master- slave architecture, in which the platform is the master node and the concrete services are located in slave nodes. For the platform system we have front-end and back-end. The front-end follows MVC model, and the backend is based on mysql. The controller of the system connects the services and manages the workflow, and the workflows execution follows pipe-and-filter design pattern. We have implemented the system in a SOA-RA way.

The overall system design is shown in following figure:



We have the OpenNEX front-end designed upon play framework, using java and scala, which supports RESTful API and read/write data in json format. The front-end is in MVC model thus clearly distinguish the controlling parts (also the workflow scheduling functionalities) from the users’ view. Observer pattern is exploited here, too.

In the OpenNEX Back-end, we have the database based on Mysql, The authentication and workflow mementos are recorded there. Here we implement memento pattern to record the execution status of the workflow at particular time-spot, and users can query for the workflow status easily from the database.

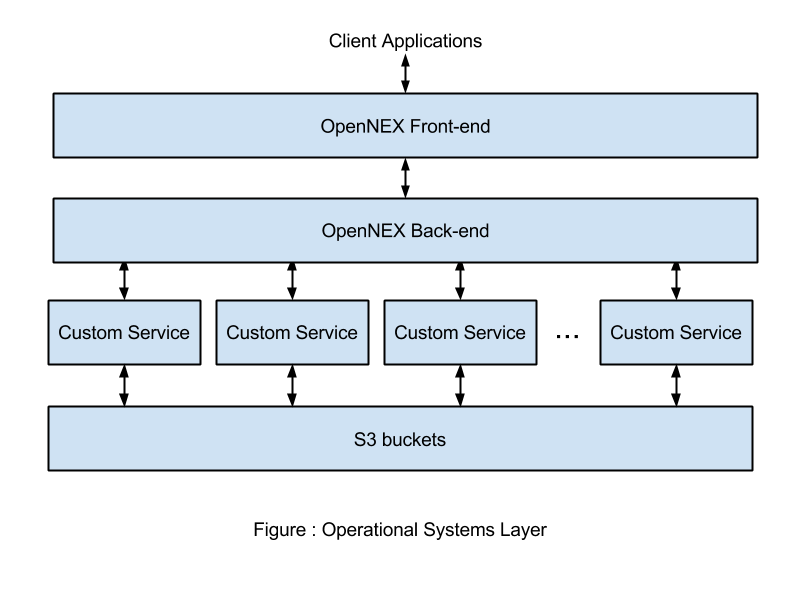
The Back-end also contains the system controller which schedules the workflows concurrently, where the workflows are executed on multiple nodes and connected through RESTful API.

The Custom Services are the concrete services that make up the workflow. Each services are recorded in flyweight format, just the id, description and URL. So flyweight pattern is exploited here for easier management of the services and better reuse of the services. It saves much of the memory capacity to support the system, because we are doing any concrete services here, and we only need to record very little information, just the id and URL to access a certain service.

Each service can be run on completed irrelevant nodes, Amazon EC2 as an example, and the controlling system needs to manage the URLs of each service. The services can be reused within each workflow or across different workflows. The probability of reuse is very important, because certain essential tasks, like sorting, could be used multiple times even in one workflow but for different stages. One important concept here is the master-slave architecture, which is a compulsory if the system needs a centralized architecture but when the scale grows to certain degree, the scheduling and false-tolerant mechanism is an essential issue.

Here we show the system architecture by borrowing the SOA-RA layers to explain architecture in detail:

## Operational Systems Layer

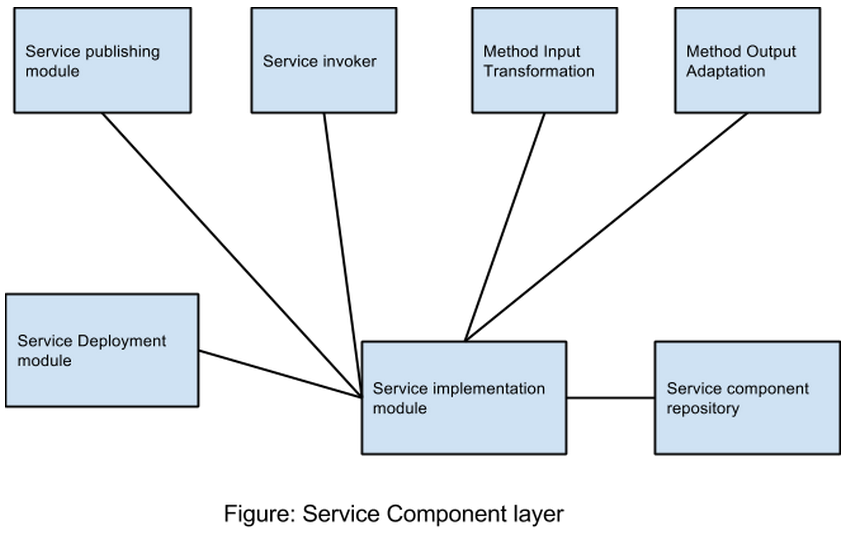


The Operational Systems layer describes the runtime and deployment infrastructure; the programs, platforms, application servers, containers, runtime environments, packaged applications, virtual machines, etc. that are on the hardware and are needed to support the solution.

NEX uses many IaaS solutions from Amazon Web Services, such as Elastic Compute Cloud (EC2), Simple Storage Service (S3), and Elastic Load Balancer (ELB). EC2 provides the raw computing power in the form of provisionable virtual machine instances. All software components, runtime environments, as well as many supporting tools reside inside the instances. NEX groups together the EC2 instances behind an ELB, which has an auto-scaling feature. This allows dynamic upscaling/downscaling the computing power according to various metrics, such as CPU usage, network bandwidth usage, or the number of running processes.

All datasets, workflows, algorithms, and any shareable data are stored in S3 buckets. The buckets are accessible by EC2 instances to retrieve and store data. VM images can also be stored on S3 to support sharing common tools pre installed on a VM image.

## Services Component Layer



This part of the subsystem is related to creating the workflows that can be easily executable by fellow scientists and users. On the professor’s suggestion we decided that we will be using Taverna as the programming interface for workflow management.

The service implementation module is the Taverna software. It consists of three main components:

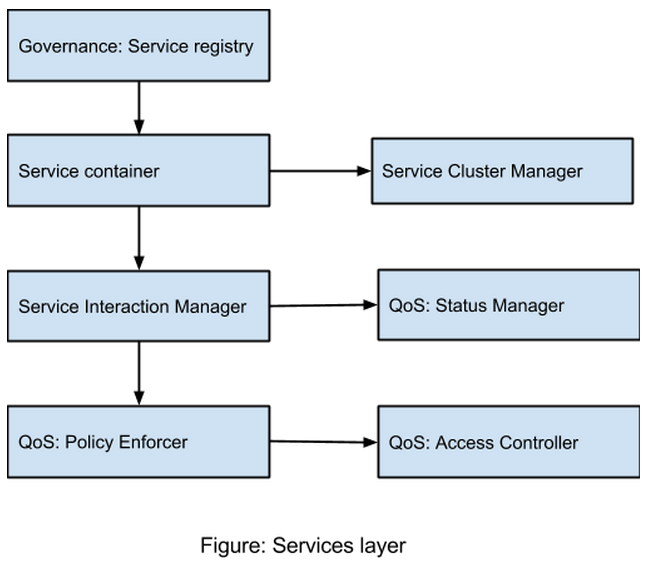
* The Taverna server where all the workflow modules are deployed and can be remotely accessed.
* The Taverna player which can be used as a:
  + REST API
  + As an embeddable web page widget
  + As a web interface in a browser
* The Taverna workbench which allows scientist and users alike to create, edit, run, delete, close workflows.

The Taverna server can be deployed using the service deployment module working in sync with the modules in the operational system layer.

The Taverna workbench enables us to work with published and invoke those services. It allows us to specify the input and output data required and helps in connecting the components in the workflow together with each other using input and output ports.

The Taverna player is accessible to the services layer to run the workflows as a REST API or as a web interface based on the specific requirements of the users.

## Services Layer



The services layer provides two types of services to the users:

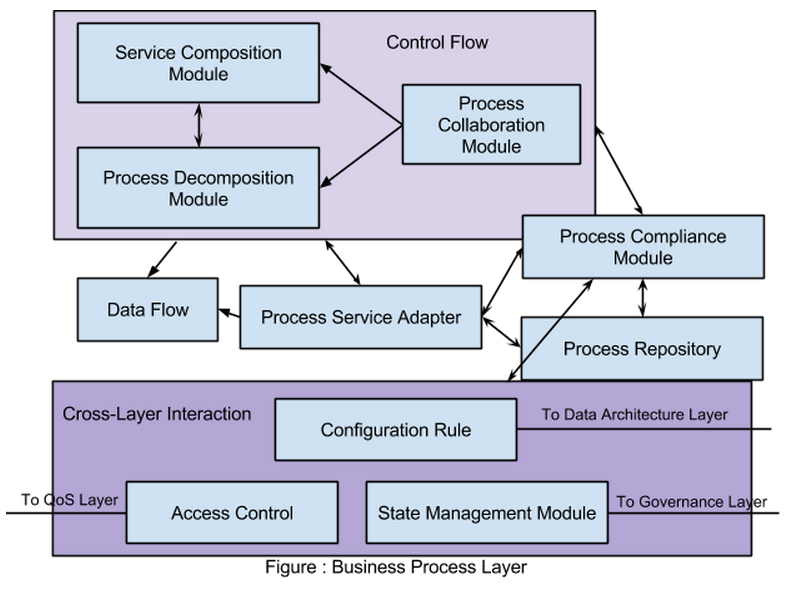
* Create or update existing workflows- Workbench
* Run and share existing workflows- Taverna player

Users who want to create workflows are given appropriate accesses to the available published services via Taverna workbench. The service container along with the cluster manager make this possible. They provide a user with a virtual environment to work on creation of his workflow.

The service interaction manager makes sure to make all the existing workflows available to the user via an appropriate web interface/portal. The workflows can then be run using Taverna players REST API which can be used by installing the corresponding ruby gem.

As an extension to existing services provided by Taverna player, we provide the user to share the workflows using the scientific social network sub system. The QoS layer helps in enforcing any security policies and provide access to users to the workflows based on NEX access control lists. The status manager make sure to give feedback on the status of different runs of the workflows with appropriate warnings and/or errors.

## Business Process Layer



The business process layer is responsible for leveraging the Service layer to quickly compose and choreograph services and to coordinate business processes to fulfill customer requirements. Business process layer make uses of the Service and Service Component layers, manages them to realize the business process goal.

As shown in the Figure above, the Control Flow module is responsible for the control on building up the business process component and manage the concurrency and collaboration of them. In the NEX project, we need to have direct control on the workflows that we currently possess, and the main business processes are the user-defined workflows, so the the Control Flow module is of core significance.

The Control Flow module consists of three building blocks:

* The Service Composition Module is for the bottom-up stream that builds the business process using current service blocks.
* The Process Decomposition Module is for the top-down stream that decomposes the business process into different service clusters that handle different functionality parts.
* The Process Collaboration Module defines how the business processes share the service blocks and how the collaboration between processes and services happens.

The Cross-Layer Interaction module defines the functional parts that overlap greatly with other layers. In the NEX project, the Cross-Layer Interactions provide essential feedback so that the stakeholders and users can optimize the workflow according to the feedback information from different layers.

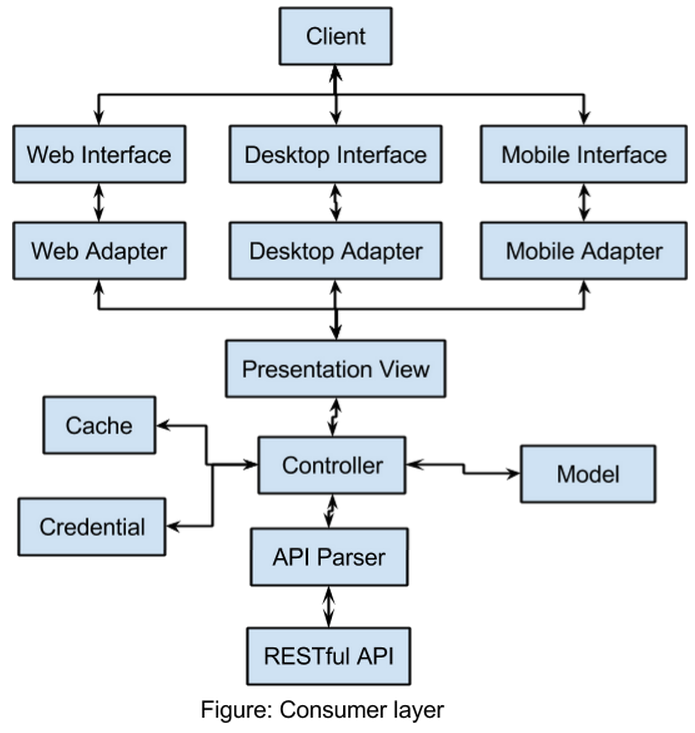
The Cross-Layer Interaction module consists of three building blocks:

* Configuration Rule module defines how each architecture building block in the figure can be configured at runtime. In
* Access Control module is responsible for the authentication and authorization of each business process and the services that it includes.
* State Management module is for recording the configuration process and system state so that users and stakeholders can keep track of the state of system.

There are four more building blocks that help the NEX project manage the submitted workflows:

* Data Flow is for managing the data flow between processes and services, including the data streaming and transaction management.
* Process Service Adapter is for adapting the business process to a service API, so that outside the module users can invoke a business process in the same way they invoke a service.
* Process Compliance Module is for checking the business process’s mechanism & result is compliant with the intention of the user/stakeholder that submit the business process.
* Process Repository module acts as a cache to the retrievable process pool that contains all the current available workflows, which user/stakeholder can use to search for workflow patterns.

## Consumer Layer

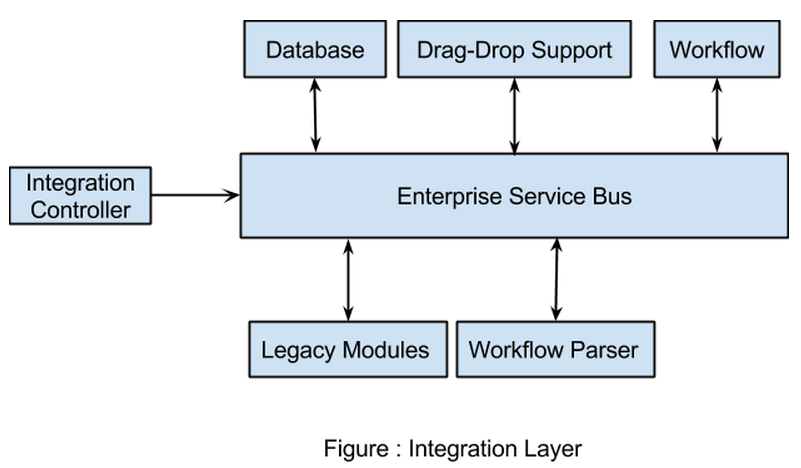


The Consumer Layer is the point where consumers interact with the SOA. It enables an SOA to support a client-independent, channel-agnostic set of functionality, which is separately consumed and rendered through one or more channels (client platforms and devices). Thus, it is the point of entry for interactive consumers (humans and other applications/systems) and services from external sources (e.g., Business-to-Business (B2B) scenarios).

NEX is a scientist-oriented software, which means the users may not be a computer genius. So friendly user interface and all platforms supported applications are required. The application needs to support the basic need of scientist users, for example, workflow execution, workflow design, upload/download, progress monitor, and so on.

In order to simplify the design of the services layer, the NEX Consumer layer reuses the same presentation view via introducing different view adapters for different platforms. For the basic client side application, MVC is the fundamental architecture style. Together with the help of local cache, MVC can take advantages of the service via RESTful API.

## Integration Layer

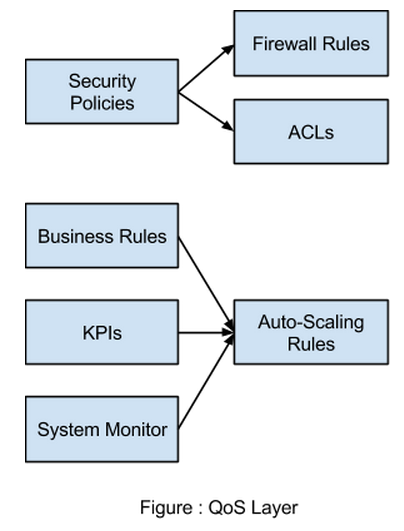


The Integration Layer is a key enabler for an SOA as it provides the capability to mediate which includes transformation, routing, and protocol conversion to transport service requests from the service requester to the correct service provider. Thus, it supports the capabilities required for enabling SOA such as routing, protocol support and conversion, messaging/interaction style, support for heterogeneous environment, adapters, service interaction, service enablement, service virtualization, service messaging, message processing, and transformation.

NEX is heavily based on service modules. Since all scientist users’ softwares and workflows are encapsulated as services. An Enterprise Service Bus is needed to connect all the workflows and also system infrastructures.

For example, user are supposed to use Drag-Drop to modify and reuse the workflows, in order to support this function, the system needs to provide both Drag-Drop lib and workflow parser, which can not directly provided by the RESTful API between Consumer and Service. That’s where ESB comes in place.

## QoS Layer

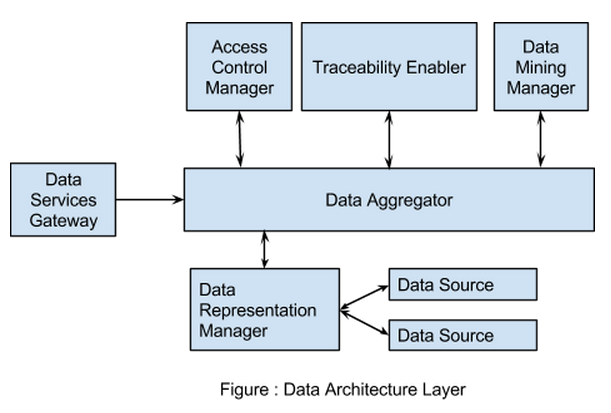


This layer provides solution QoS management of various aspects, such as availability, reliability, security, and safety as well as mechanisms to support, track, monitor, and manage solution QoS control.

The security aspects are handled by firewall rules and access control lists, both are derived from security policies defined by NASA. The firewall rules will restrict inbound/outbound connections to make sure that only certain ports are open to certain applications. Access Control List (ACL) is a list of permissions attached to a resource, such as datasets, algorithms, workflows, etc.. An ACL specifies which users or system processes are granted access to resources, as well as what operations (share/unshare) are allowed on given resources.

This layer also provides the ability to manage virtualization of computing resources through auto-scaling rules. The rules are derived from business rules, Key Performance Indicators (KPIs), and metrics from system monitor. In case a cluster node is down, the event will be captured by system monitor which will trigger a new VM instantiation to cover the failing node. The upscaling/downscaling rules are also influenced by KPI so that the entire NEX system always meet the standard quality of service.

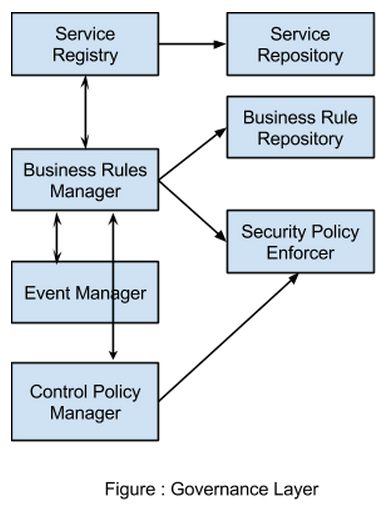
## Data Architecture Layer



Data Architecture Layer is responsible for unified representation of data as well as data consistency and data quality. Other responsibilities of this layer are business analytics, business intelligence, data virtualization, metadata considerations and controlling the access to data. Data Architecture layer provide the basis for all the aspects of data and business intelligence schema that are necessary for the realization of services all the way down to the databases.

* Data Representation Manager hides various data sources and present data in uniform formats to the requestors. This ABB manages the relationships of different data sources and virtualizes the data in order to present it in a standard format.
  + Even though a data from different data sources may have different units or data types. With data virtualization, data architecture layer can transform the data and makes sure every requestor is being served in a standard format.
* Access Control Manager handles access privileges of users to data and controls access on individual data elements.
  + NEX has many user tiers and every user has different levels of access to data. This ABB can regulate the access either by denying or accepting the access request. It can also make this layer to serve partially obfuscated/noised data for privacy reasons (conditionally accepting the request).
* Data Aggregator dispatches requests to other ABBs in this layer, handles data transformation and compiles information from different data sources or components with intent to prepare combined datasets for data processing.
  + NEX
* Traceability enabler monitors, manages and logs who has accessed the data, when, and what part of the data has been accessed.
  + NEX has many resources, tracing logs can be used for business intelligence and auditing as well as billing for resource usage (like computing).
* Data Mining Manager is responsible for analyzing data access logs and providing optimization algorithms and business intelligence for data optimization. It enhances business intelligence by enabling the discovery of previously unknown trends and patterns in the enterprise data.
  + Depending on NEX tracer logs, certain trends and patterns can be revealed which for example enables smart caching of data or help NEX adjust its cloud resources by scaling according to date and time.
* Data services gateway supports interfacing between the Data Architecture Layer and the consumers of this layer's services. This ABB provides a consistent entry point to the Data Architecture Layer through multiple mechanisms and is critical to expose Information as a Service (IaaS).
  + A NEX IaaS requestor may have multiple mechanisms such as messaging, service calls, and batch processing. Data services gateway handles multiple mechanisms and act as a single point of contact with the rest of the data architecture layer.

## Governance Layer

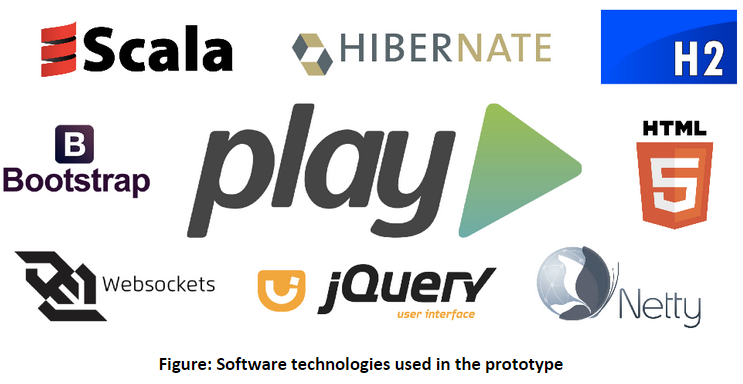


The governance layer is mainly to enforce the conformity of the system and ensures the system adheres to defined policies and standards. It monitors and reports the health of SOA services and processes. For NEX project, the main focus of governance layer is to keep track of the health of workflows and ensures the access control, security and QoS of each process is within healthy status.

* Services register through Service Registry module and are then saved in Service Repository which is a retrievable service/process pool where user/stakeholder can search for available usable process/service or search for similar service patterns.
* Business Rules Manager module monitors and reports the status of business process in terms of their compliance with defined business models and business rules. The business rules as the regulation of this system are saved in Business Rule Repository.
* Security Policy Enforcer enforces the compliance of security policies in the whole system. All the necessary authentication and authorization process should be done in a secure manner according to the Policy Enforcer.
* Event Manager manages the events and exceptions in the whole system and reports them to the Business Rules Manager to renew the health status of each business process (workflow).
* Control Policy Manager manages the changes of authentication/authorization procedures according to the changed control policy, and configures the Security Policy Enforcer to be compliant with the new control policy. It also monitors each business process/service and reports their policy-compliant statuses to the Business Rules Manager.

# System Implementation

We have implemented a working prototype which covers the Service Sharing and Workflow Sharing capabilities. The prototype is built using the industry-standard Play Framework for Java, along with other supporting software technologies which forms the full software stack.



Play Framework is the leading standard for developing scalable, production-ready web applications. The framework itself comes with two flavor: play-java and play-scala, which uses Java and Scala as the programming language, respectively. While most of the features are available through on both flavors, the latter has more flexibility and supports Scala-specific features (e.g. implicits). That said, for this project, we are using play-java to maintain compatibility and faster time to develop.

Play Framework provides an MVC-style separation between software component layers. It also has the capability to handle URL routing so that URLs can be mapped to a corresponding Controller action. It also offers an easy session management (e.g. cookies, login sessions, etc.) out of the box. The View layer is handled by Scala Template, an HTML templating engine which is capable to render data passed by a Controller to fragments of HTML file. The Model layer is handled by Hibernate, an ORM engine which can manage the mappings between several entities in a database. By using Hibernate, all related entities (connected by OneToMany, ManyToOne, and ManyToMany relationships) will also be pulled from the database when we are querying for a particular entity.

To make the prototype easier to deploy and test, we are using the built-in H2-mem as the RDBMS, which stores all data on memory. This will allow fast setup without having to install a separate database system. All HTTP requests to the application are served by the embedded Netty HTTP server on port 9000. Websockets are also used in this project for the output data of a workflow execution to be displayed real-time on the client’s browser. All of the UI components are provided by Bootstrap and jQuery UI.

The summary of software technologies used in this prototype along with their purposes are described as follow:

|  |  |
| --- | --- |
| **Technology** | **Purpose** |
| Play Framework - Java | Base MVC framework, URL routing, actions, session management |
| Scala Template | HTML templating engine |
| Hibernate | Object-relational mapping |
| H2-mem | SQL database engine |
| Netty | Embeddable HTTP server |
| Websockets | Real-time client-server communication |
| Bootstrap, jQuery UI | UI components |

The setup guide for running the prototype is as follows:

* Install:
  + Download Play Framework 2.3.x from its website
  + Extract zip file then add activator to $PATH
  + Clone the project from this github repo
* Build:
  + Open terminal on the local project directory
  + $ activator clean compile
* Run:
  + Open terminal on the local project directory
  + $ activator run
  + Browse http://localhost:9000 with your favorite browser (except IE)
  + Select apply evolutions to initialize the database

Apply Evolutions will initialize the database with the necessary tables and keys.

User authentication and access control are handled by the Security.Authenticated class from the Play Framework. It acts as a request filter that can match the request header with browser session to see whether the user is authenticated or not.

Services are just a regular REST web service which accepts a JSON as input and will also output a JSON with the same format. The JSON format is as follows:

|  |
| --- |
| {  “content”: (base-64 encoded plain text)  } |

Base-64 encoding is used to encode newlines and any other characters outside the ASCII specification.

Example web services are available in the application for testing purposes:

|  |  |  |
| --- | --- | --- |
| **Web Service** | **URL** | **Purpose** |
| Tokenize | http://localhost:9000/api/v1/tokenize | Split input string by whitespace chars |
| Sort | http://localhost:9000/api/v1/sort | Sort, just like in UNIX |
| Map | http://localhost:9000/api/v1/map | Sample mapper for wordcount example |
| Reduce | http://localhost:9000/api/v1/reduce | Sample reducer for wordcount example |

One then can create a Wordcount example workflow for testing the services by connecting Tokenize → Map → Sort → Reduce by using the drag-and-drop UI on the Create New Workflow screen.

Notifications will be sent to the users of a service if the service got changed by another user. This is done by iterating all the users registered to the service.

# Experiments and Analysis

We created four kinds of services for testing (map, tokenize, sort, reduce), the service

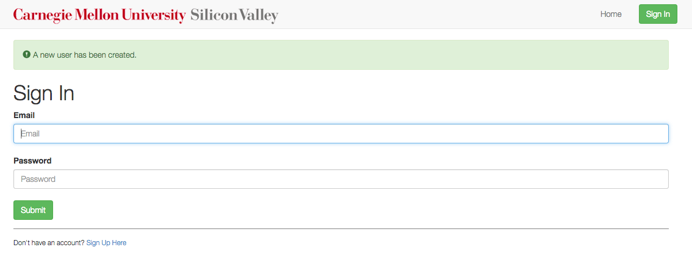
All services can run on different machines, providing their URLs, use Restful API to pass the input/ output data. Services can be reused between workflows and within each workflow (invokes one service multiple times). Workflows and services can be shared between users.

For the UI testing, we can go through the whole process to test the system:

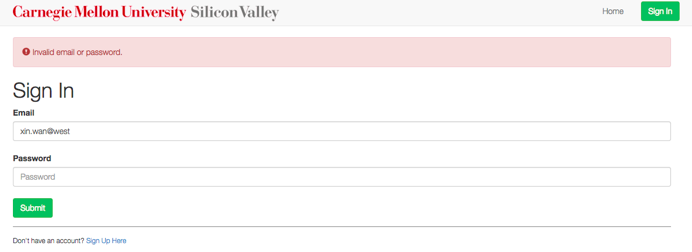
First, we sign up an account:



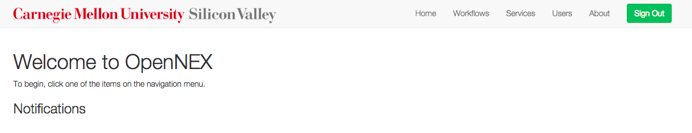
Sign up successful!



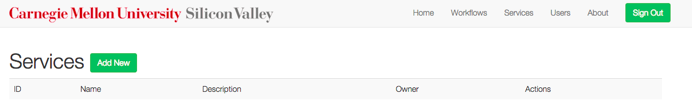
Then, we sign in the account with the wrong password, we will get following error message:



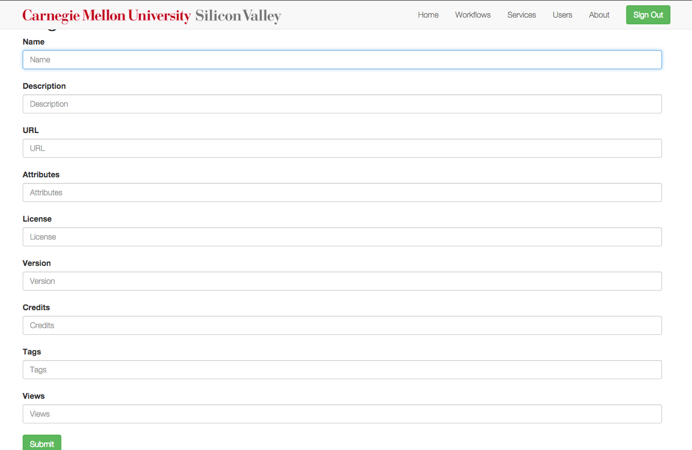
Then we log in with the correct password (note that there is no notification now):



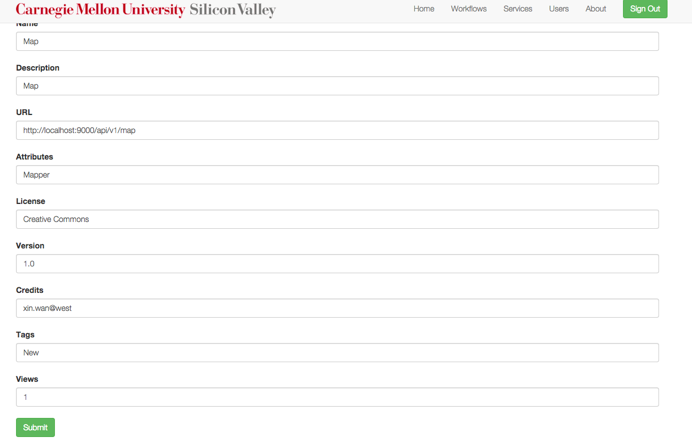
We switch to services part:



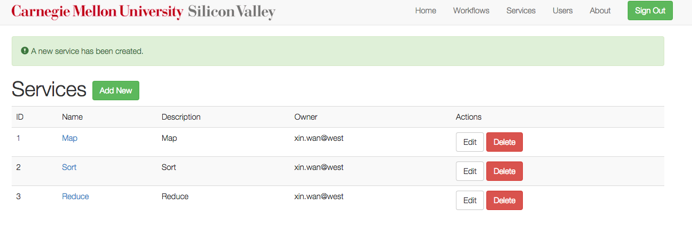
For the execution testing, we can take a classic work count example to demo the workflow part. So we will register three services: map, sort and reduce. We click Add new in the services interface, we get following page:



We register the map service by providing the corresponding URL (here is <http://localhost:9000/api/v1/map>):



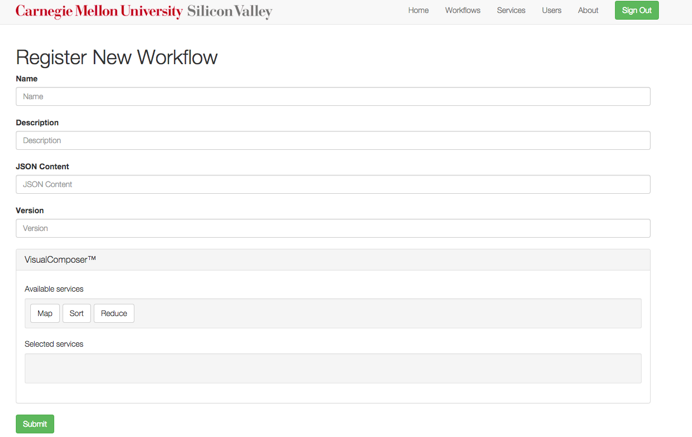
We can then create sort and reduce services in the same way, and we get three services:



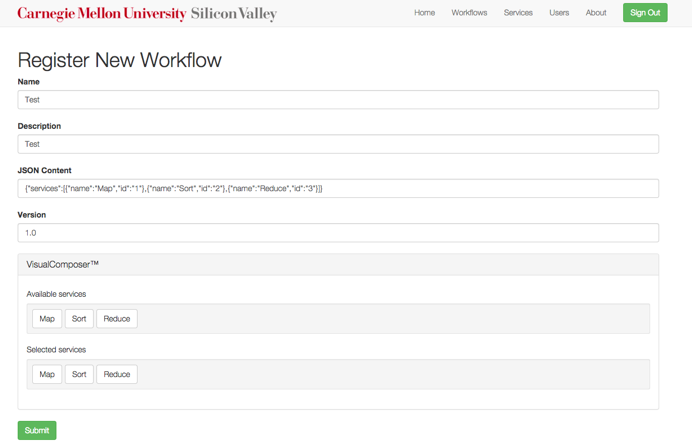
Then we switch to the workflow page:



We click Add New, then we have this register new workflow page, as shown in the figure, our implementation supports drag-and-drop functionality to easily drag and drop registered services to the current workflow to form a service sequence. Moreover, each service can be reused many times within one workflow.

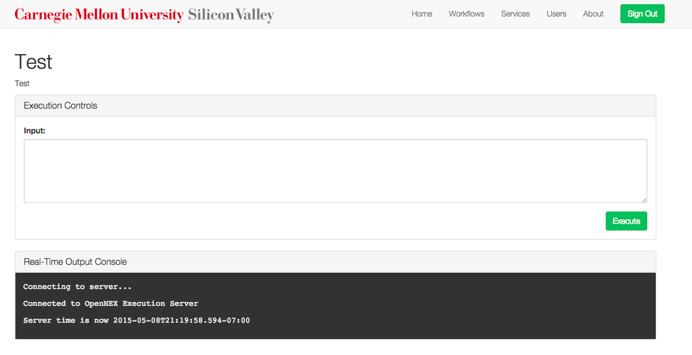


Here we register the simple word count example:

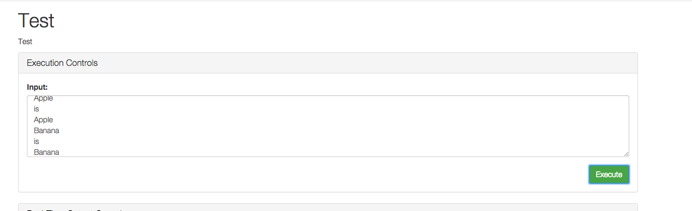




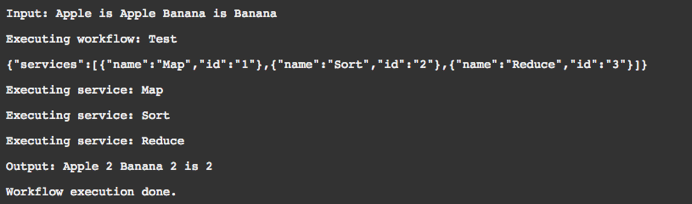
We click execute and enter this execution page:



We enter some input:

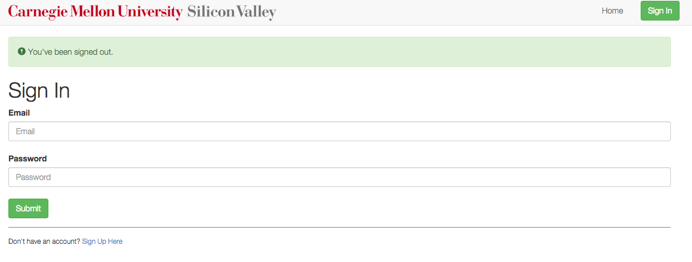


And the workflow outputs the word counts:



We can rerun the workflow with different input, and we can resume the workflow whenever we want to as long as it is registered.

We can sign out by clicking the sign out button on the top right corner.



Moreover, our system supports notifications for the needs of broadcasting or informing users about the module changes. Here for example, if some users edited the registered workflows or services, other users will be notified in the notification column of the Home page. They will see the notifications stating that certain services are changed just as following figure shows:



## System Design patterns implemented

The Front-end system is based on play framework, follows MVC pattern. Throughout the project we’ve implemented many design patterns such as Pipe & Filter, Publisher Subscriber, Observer, Flyweight, Memento, Command etc.. The brief introduction of the design patterns we’ve used is shown in the following table:

|  |  |
| --- | --- |
| **Pattern** | **Purpose** |
| Pipe & Filter | For services concatenation within workflow |
| Publisher Subscriber | Publishing & Receiving notifications |
| Observer | Receiving notifications |
| Flyweight | Reuse services |
| Memento | Save the workflow execution status |
| Command | Encapsulate the workflow as an object |

# Future Work and Conclusion

At the this point, our system consists of the core functionality required for the proof-of-concept platform and open to further improvement. By tackling low hanging fruits on the short term, and focus on more complex functionality in the long term can conserve the motivation and the momentum of the project. For example, currently our system only supports linear workflows. One of the short term improvements that can be implemented in a short period of time is workflow interface that requires multi-linear workflows. This UI feature can easily be implemented using open source jsPlumb library which is available on github.

Another short term improvement is data visualization services. We can use open source D3.js library available on github for easily developing this service UI and introduce more features over time.

Long term improvements that we foresee are managing cloud resources for the user and creating a learning portal for attracting new users. These improvements involve more work and risk but they are very important for increasing the user base and engaging students. If we can manage cloud resources for the user and introduce a good UI, we are removing this burden from the user's' shoulders and letting them focus only to the data. Adapters for various cloud providers are needed like Google Cloud Platform, Amazon Web Services, Microsoft Azure and rackspace. We must develop and maintain these adapters, also managing the user credentials are a sensitive security issue which we must be dealing with.

Creating and maintaining a quality learning portal consisting of tutorials, step by step videos and supplementary material requires time and effort. However the presence of such a service is key to engage new users and young students that may become successful scientists of the future. By allowing more users to access the publicly available data, we are increasing the value that we can generate out of the available data, which is one of the key motivations for OpenNEX platform.

During a one month period, we designed and implemented a proof of concept Service Oriented Architecture for OpenNEX in such a way that it would be extensible and could easily be integrated with other development efforts from other teams. We tried to enhance interoperability and usability with a focus on modularity. We are excited and curious for the future of OpenNEX and the value that users will bring.