Distributed Social Networking

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Crystal Dulay
Andrey Gusev
Arash Motamedi

December 2010

APPROVED FOR THE COLLEGE OF ENGINEERING

John Gash, Project Advisor

Dan Harkey, Director, MS Software Engineering

Dr. Sigurd Meldal, Chair, Computer Engineering Department

ABSTRACT

Distributed Social Networking
By Crystal Dulay, Andrey Gusev, Arash Motamedi

Virtual and social networking services constitute a considerable and dynamic proportion of the web. Internet users participate actively in generating social content through services such as Facebook, Twitter, LinkedIn, and numerous other services, as well as personal websites, blogs, and online forums. Today, each person's virtual identity portrait is comprised of many discrete profiles, contents, and social links hosted on disengaged service providers that often carry redundant, out-dated, and at times, contradicting data. Due to the disjoint and varying information sharing policies, web search engines yield inaccurate and unreliable data about a particular person being searched for.

The vision for this project is to create a person-centric, as opposed to website-centric, distributed social networking service. Areas of interest include attracting and managing a large user base and addressing architectural aspects such as scalability, reliability, performance and connectivity between different social networking sites. Other areas to explore are the legal and privacy concerns to securely aggregate and store user data retrieved from multiple sources. Additionally, the use of data mining techniques will allow for the discovery of user interests and the suggestions for potential social introductions.

Design and development of this project is a step toward realization of Web 3.0 and semantic web through implementation of a clustered, social-correlating service. By bridging the gap between the data islands that belong to various social networking and virtual society services, this project will act as a central gateway to each internet user's online profile and interactions. The scope of this project is to create such an infrastructure that can be utilized for attaining social and commercial goals with the prospect of becoming a major contribution to the field of distributed social networking.

Acknowledgments

The authors are deeply indebted to Professor Dan Harkey, Mr. John Gash, and Assistant Professor Magdalini Erinaki for their invaluable comments and assistance in the preparation of this study.

Table of Contents

List of Figures

List of Tables

Chapter 1. Project Overview

Introduction

The internet and the World Wide Web have been primarily used as tools for information sharing across the globe. Early use of internet concentrated heavily on producing and sharing a large amount of textual and visual content that took scientific articles, news, technical references, and books across the borders to virtually every corner of the world. With the emergence of dynamic websites—active server pages, CGI scripts, and other server-side programming technologies—interactive and customized content could be delivered to internet users. Search engines, web based email clients, online stores and forums soon became the new hype of the web.

With an ever increasing expansion and penetration rate, and a steadily declining cost to the customers, internet and web access became popular and affordable. Operating systems introduced built in support for connection to the internet and provided browsers for viewing content and interacting with the web. Web access was no longer a luxury, unique to the corporate or the elite. Soon, individuals started creating personal websites where they shared information of their own interest. Blogs were the next big bang of the web and started an era of personal presence on the internet.

With further expansion and increased popularity, and millions of users seeking ways to present their interests, opinions, and activities, it became increasingly likely for every individual to find traces of their friends, family and acquaintances on the web. And of course, the next ground breaking idea was to create a network of connections, a social network, in which every participant could share information about themselves, while maintaining contact with their circle of friends or links. In a short amount of time, social networks emerged in all shapes and flavors. In recent years, the popularity of social networks has grown steadily, and having an account on one or more social networking websites has become as essential as having an email address.

Many internet service providers have been in constant competition to provide their users with a similar set of features and services. Internet giants such as Google, Yahoo and Microsoft not only offer the more traditional email and search features, but also directly, or through their affiliate services, provide users with means for writing blogs, sharing pictures and videos, creating personal web pages, and connecting and keeping up with their friends. For instance, Orkut (a Google service), Yahoo 360, and Microsoft Live Spaces are essentially the same social networking services with largely similar features and possibilities wrapped up in different shapes and presentations. Competing with such major players are websites such as Facebook, Twitter and LinkedIn—and Orkut, MySpace, Hi5 and Tagged among older services—that were launched with *Social Networking* as their specific and narrow area of focus.

Many of the aforementioned services provide overlapping features. For example, Yahoo (Flickr), Google (Picasa Web Albums), Microsoft (Live Spaces), Facebook and Orkut provide their users with a significant amount of space and a convenient set of tools

for uploading, hosting, tagging, sharing, and commenting on photos. Or, Google (YouTube), Facebook, and MySpace let their users upload and share video content. And, virtually all of the services mentioned above, let their users “add friends” and stay in touch with them. However, each of these service providers has been able to promote one or a few of their features as their primary and most successful and distinguished service. That is, YouTube, Flickr, and Facebook have branded themselves as the de facto standard services for sharing videos and photos, and maintaining online social connections respectively.

Close to two billion people, 25% of world’s population, access the internet and some of the internet service providers listed above have anywhere between tens of millions to about a billion registered users, many of which loyal to the services they have used for a number of years due to convenience and familiarity. With such significant user bases, many service providers have shifted focus from competing with each other for providing the same or similar services, to exploring new grounds and creating bridges across currently existing services regardless of provider. Google has recently introduced its Buzz service, which retrieves information about a specific user from a variety of sources. Microsoft Live Spaces lets users list their “web activity” from over 50 sources, including Twitter, Facebook, and YouTube.

The movement toward creation of person-centric networks (as opposed to feature, or website-centric networks) can also be visibly witnessed on cell phones and smart phones. Some Android phones come with a service called FriendStream, which keeps the subscriber posted of all updates their friends make on various social networking sites. New Palm handsets sync their phonebooks with the user’s Facebook, Google, and Yahoo accounts to create an integrated contacts management system. Windows 7 Mobile has a feature to let the user update their status on multiple social networking websites simultaneously.

Distributed social networks are being welcomed and extensively adopted by many websites. New protocols and cooperative efforts among various service providers let them remain in the scene while collaborating with each other through standardized authorization interfaces and well structured public API’s. As such, each service will have the chance to continue focusing on the narrower areas where they have already excelled, while also reducing the risk of being swept away by a superior service because of the presence and the mutual dependencies they have created in the distributed social networking.

Distributed social networking is a relatively new but fast developing concept and practice. While the focus has so far been on including as many services as possible, challenges such as presenting the sizeable amount of data in a streamlined interface, and questions such as how the data gathered from multiple sources can be mined and utilized to the benefit of the users as well as the businesses providing it remain to be addressed.

Proposed Areas of Study and Academic Contribution

The current market trend and advancements in distributed social networks have been of great influence in this project's focus. In addition to providing a distributed social network, this project provides a subset of the unique services that it connects to, such as access to Twitter's user generated instant information—tweets, and LinkedIn's services of connecting employers and professionals.

This project will integrate existing platforms such as Twitter, Facebook and LinkedIn and perform data extraction and mining, for uniform presentation and as feed into the designed intelligence engine which provides analytical data as well as potential social connections and suggestions. This data will be stored locally on databases relying on distributed storage systems as described in the following sections. The authentication mechanism used for accessing users' information from other providers will be utilize the OAuth protocol as it has become a standard in the authentication and authorization process among services and is becoming more commonly adopted. Authentication by OAuth enables a standard approach without having to utilize service provider specific libraries which may cause dependency issues.

As users set up their accounts on the Distributed Social Network, they will need to provide their credentials for the supported third party service providers, which after going through the OAuth authentication flow return an access token granting access to the API. OAuth parameters are passed in the HTTP Authorization header, as the POST request body, or in the query part of the URLs. OAuth authentication is an important step as it enables secure connection and data transfer to and from service providers.

Data returned from service providers are in the form of XML documents, the structures of which vary amongst service providers. A standard data structure will be implemented to ensure that the local data can be uniformly retrieved and referenced across different service providers. An XML parser will be used to extract the data from the documents, bind the schema, and unmarshal the documents into Java transfer objects, which can be passed on to the storage and intelligence components for persistence and processing purposes.

The Intelligence Engine consists of various text analysis tools and indexing mechanisms that operate on the data aggregated from various service providers and allow for the discovery of users' areas of interests and overall trending topics. Additionally, the generation of statistics and reports on users' usage and interests provides data for suggesting potential social introductions. Text Data and Mining (TDM) and Text Knowledge (TKM) methods and algorithms will play an integral role in the data analysis processes to meet challenges such as varying languages, word root detection, and concept extraction.

The implementation of Distributed Hash Tables (DHT) will support the potentially large scale and high availability data. The DHT System will be implemented to support fast data retrieval for transactional as well as indexing and data mining

processes. A DHT System is an ideal choice due to the scalability pressures as user base and application usage grow.

Current technologies and researches provide the tools and the concepts that enable creation of a foundation to support a Distributed Social Network. The Academic Contributions are the complete aggregation of the aforementioned current technologies in a standardized approach, as opposed to mere reliance on third-party specific components and processes.

Current State of the Art

In recent years, the popularity of social networking sites has experienced an ever increasing growth. According to Hitwise, Facebook replaced Google as the most visited website in the United States in March 2010 [1]. With customers having access to many social networking sites with unique services and attributes, attention seems to be shifting toward integration of these diverse services and creation of distributed social networks.

1.1 Google Buzz

Google released Google Buzz in February 2010 as a social networking and messaging tool which integrates existing platforms such as Twitter, YouTube, and Flickr. Google Buzz allows users to share content such as text, links, photos and videos and also view and comment on other users' content. The service also provides mobile accessibility which focuses on geolocation. When accessing the service from their mobile devices, users can view and stream data not only from people they know but also other people in their vicinity, which creates opportunities for business everywhere. [2]

1.2 Security (OAuth, FOAF+SSL)

OAuth is an open protocol which allows users to implement a secure authorization API to grant a third party application access to data stored with another service provider. FOAF+SSL (Friend of a Friend, Secure Sockets Layer) is an authentication and authorization protocol that links a WebID (URI) to a public key to provide a Web of Trust (WOT) with desktop and mobile browsers. Twitter, Facebook and LinkedIn are among many social networking sites that utilize FOAF+SSL. [3]

1.3 Scalable Data Storage

Removing limitations of classical RDBMS systems, Distributed Hash Tables (DHT) are being utilized increasingly to create large scale, highly available storage systems. Within a DHT system, data is partitioned and stored on multiple servers in a redundant manner that ensures overall fault tolerance and high availability and reliability. DHTs can be extended by adding new nodes without the need to repartition the existing data. Amazon, for instance, relies on DHT to manage its massive load of data. [4]

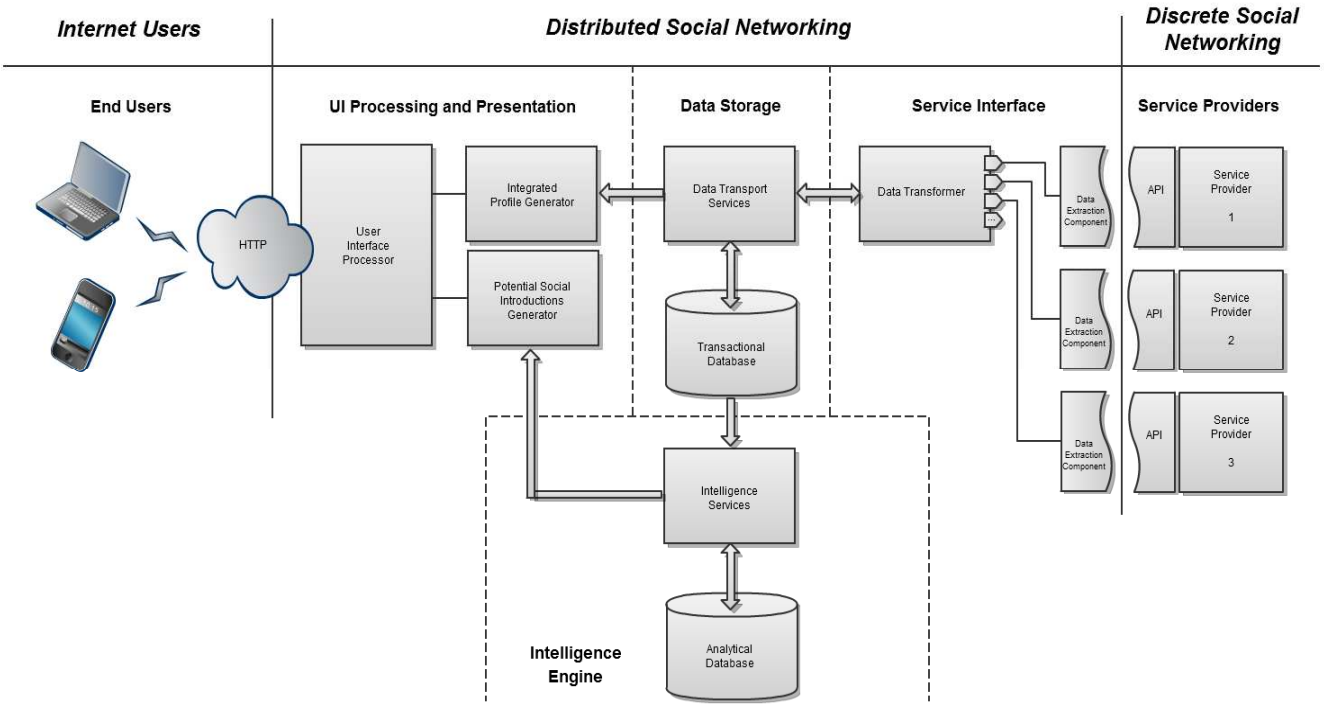
1.4 Text Analysis

Text analysis methodologies are being increasingly used by online services that generate or aggregate considerable amounts of textual data. Search engines are capable of identifying popular queries, social networking sites display hot trends based on user inputs, and news websites let their readers know what stories are buzzing. These features and services are realized through Text Data Mining (TDM) and Text Knowledge Mining (TKM) methods implemented to extract concepts from words. Once discovered, concepts and meanings concealed in sentences and phrases can be assessed for creating relevancy graphs and generating statistical data, drawing conclusions, and also producing suggestions. [5]

Chapter 2. Project Architecture

Introduction

The presented high-level architecture for the Distributed Social Networking system illustrates four major system blocks and how they fit in between the internet users on one side, and the existing social networking websites on the other. The four major system blocks include Service Interface, Data Storage, Intelligence Engine, and the UI (User Interface) Processing and Presentation.



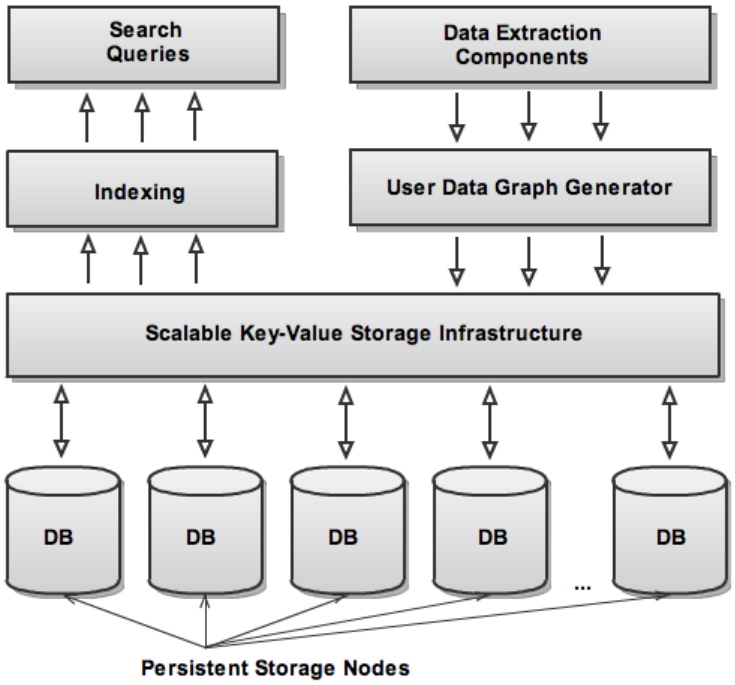
Architecture Subsystems

In order to communicate with third party social networking websites, herein referred to as Service Providers, the Service Interface block pairs the Data Transformer component with a series of Data Extraction Components that interface with APIs provided by the Service Providers as means to access and retrieve user data. The Data Extraction Components convey raw data to the Data Transformer component which is in charge of understanding, validating, filtering, conforming, merging, and reporting user profile data to the Data Storage component.

The Data Storage block is in charge of all transactional data operations and provides service to the three other components of the system. The Data Transport Services component relies on a scalable and high performance Transactional Database system to perform three major tasks: retrieve and refresh user profile data through

communication with the Service Interface, provide data feed to the Intelligence Engine, and also provide user profile data to the UI Processing and Presentation block for profile generation and display purposes.

The following diagram illustrates a high level preliminary design for the Data Storage subsystem. The Distributed Social Networking site is expected and required to handle large amounts of data that might exceed the capabilities of a single database server. Moreover, it is a requirement of this system to ensure data access reliability and fault tolerance. In order to achieve these goals, a distributed key-value storage infrastructure that uses a set of persistent nodes to create a highly scalable and fault tolerant data storage system is designed as follows. The Data Storage infrastructure will be responsible for persisting User Data Graphs and providing access to those graphs for transactional and intelligence processes.



Scalable Data Storage Design

The Intelligence Engine works alongside the Data Storage and tries to discover conceptual and meaningful connections and relationships within the massive load of user data available to the system. Paired with a capable Analytical Database, the Intelligence Service component is in charge of discovering users' areas of interest and providing analytical findings, conclusions, and suggestions to the Potential Social Introductions Generator component of the UI Processing and Presentation block.

Utilizing existing state-of-the-art text analysis tools alongside original machine learning and indexing mechanisms, the Intelligence Engine will be able to produce statistics and reports, draw conclusions, and suggest potential social and professional introductions for each system user.

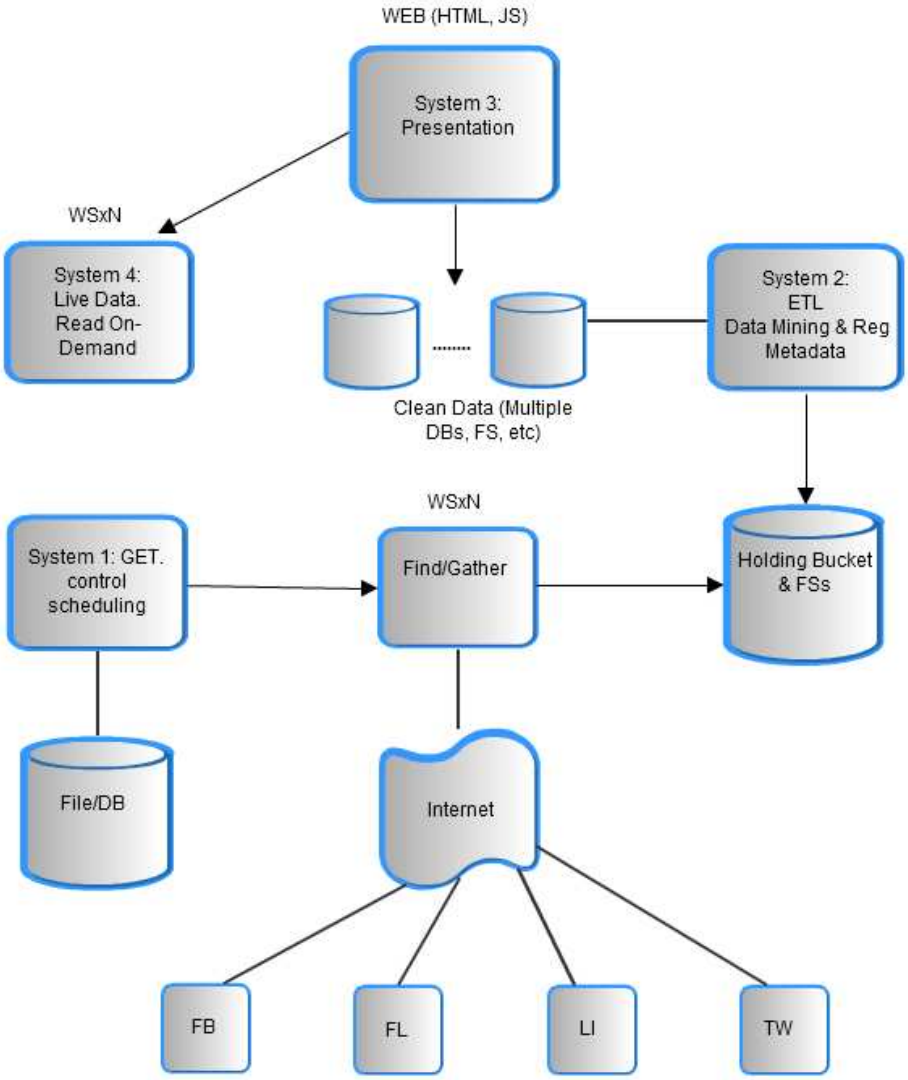
Lastly, the UI Processing and Presentation block is in charge of combining data provided by the Data Storage and the Intelligence Engine components and presenting an integrated user interface that includes user profile data along with potential social introductions. The UI Processing and Presentation block is the public interface of the Distributed Social Networking system that provides access to the users over the World Wide Web.

As the only point of interaction with the end users, the UI needs to adhere to certain design standards and principles. Since most social networking services provide a considerably similar feature set to their users, a significant and determining competitive factor among all service providers appears to be the slickness of their UI. Web 2.0 web pages are now any internet user's expectation, and this project would be no exception.

The users of the Distributed Social Networking website will not only have access to a streamlined Web 2.0 interface, but would also be able to customize and tailor the layout of their profile pages to their liking. The Distributed Social Networking website will act primarily as a data source which can be paired with any presentation and interfacing technology including web, web services, desktop and mobile applications and the like.

Intermediate Architecture

After extensive discussions and evaluation of numerous architecture alternatives, and per recommendations and instructions made by the project advisor, the following intermediate architecture design was devised, which serves as the foundation of the final project architecture upon which the system was implemented and deployed.



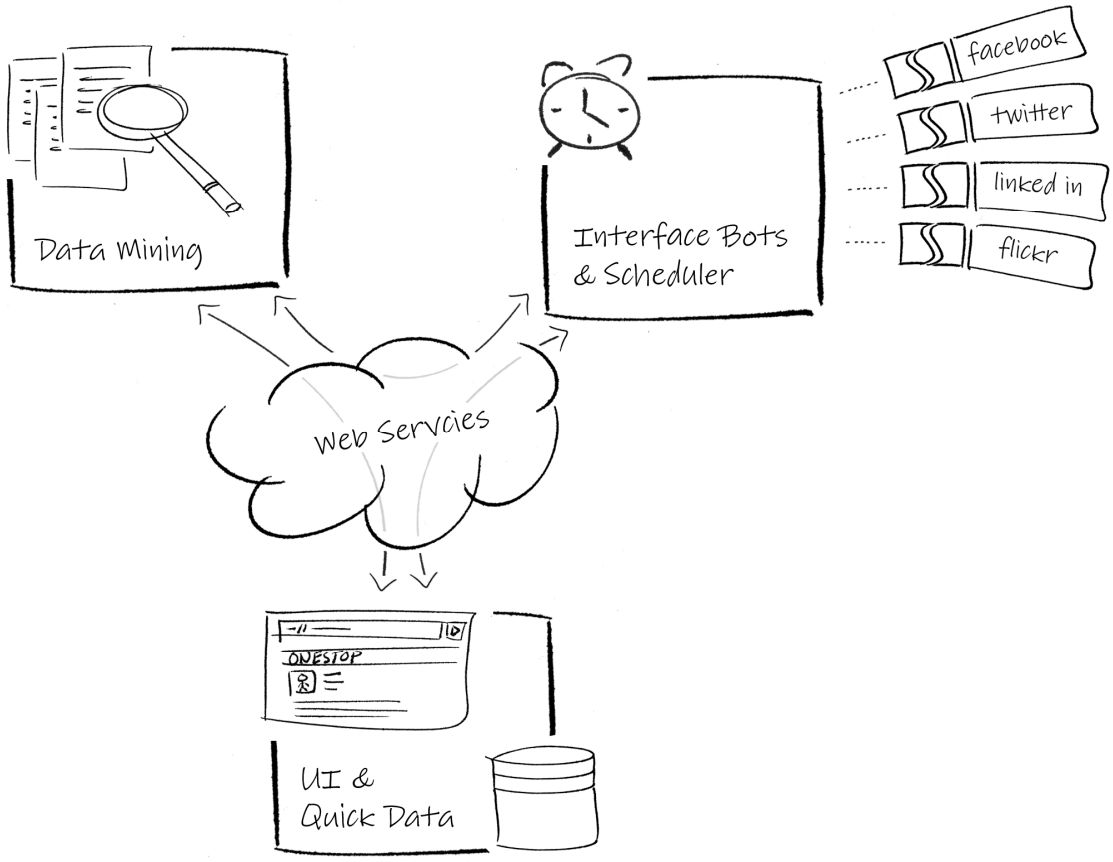
Intermediate System Architecture

Chapter 3. Technology Descriptions

The Distributed Social Networking project is an exemplary model of a distributed system in concept, design, and implementation. Diversity of software techniques and practices is at the core of the architecture and design devised for this project. In continue, the challenges faced through the course of this project along with their solutions will be presented and thoroughly discussed. The proposed solutions have come about after much contemplation, discussions, researches, and development of prototypes.

3.1. The Distributed Nature of the Project

The name "Distributed Social Networking" refers to the fact that this project interfaces with multiple social networking service providers—namely Facebook, Twitter, Linked in, and Flickr. However, the project itself has been architected, designed, and implemented to achieve a high degree of distribution, scalability, performance, and reliability. The major components illustrated in the architectural overview have each turned into a standalone system that provide services to each other in a well orchestrated manner to carry out the promised functionality of the system. It is worthy to revisit a top level overview of the major system components as they have been implemented.



High Level System Architecture—Major components and interoperability infrastructure

In continue, the technical complexity and viability of this model is discussed and the design and its implementation is justified.

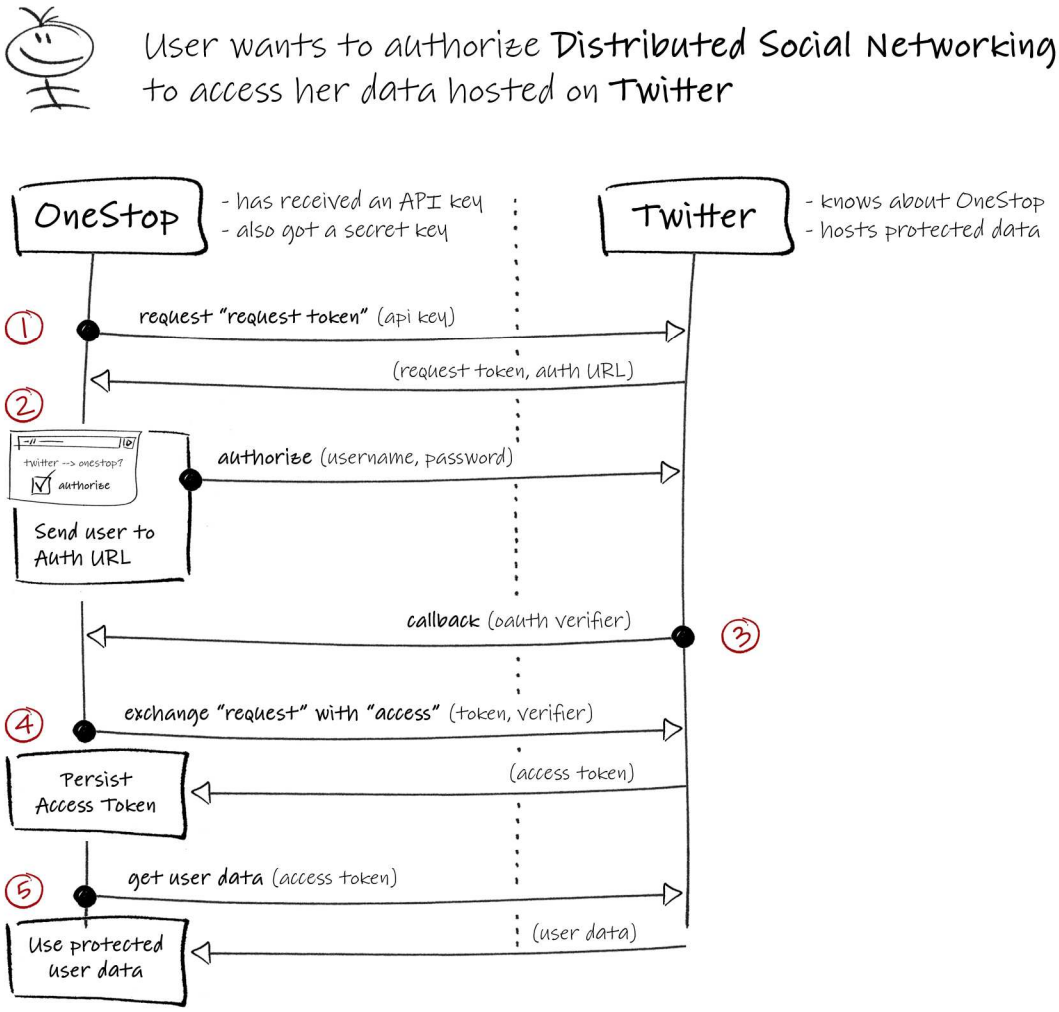
3.2. External Service Interfacing

As illustrated above, the Distributed Social Networking system interfaces with four prominent and majorly popular social networking service providers—Facebook, Twitter, Linked in, and Flickr. The reasons for choosing these services among many that exist on the internet have been discussed earlier. However, among other reasons, the attitude of these four specific service providers toward offering an open and relatively easy to utilize API is of significant weight in their choosing.

3.2.1. OAuth Authentication Model

All four chosen service providers implement APIs and interfaces that enable external services—this project for instance—to communicate with them, authenticate on behalf of their users, and retrieve and, with a higher access level, modify user data and updates. The APIs, though unique and varying across service providers, follow certain standards and distributed software practices. OAuth, as discussed earlier, provides a secure way for authenticating and authorizing users across services. The four chosen service providers each implement a particular version or flavor of the OAuth standard authentication process.

Facebook, for example, implements OAuth 2.0, Twitter and Linked in implement OAuth 1.0a, and Flickr provides an *OAuth-like* interface which is similar, though not exactly the same as OAuth 1.0 specifications. OAuth, though robust and secure, is a cumbersome authentication process which sends the users back and forth between the host and the guest service providers and requires the service providers to contact and communicate with each other multiple times to successfully finish the authentication and authorization process. The following diagram might help understand the procedure. Please note that the Guest Service Provider, on the left, is the service provider that needs to gain access to user accounts and data hosted on the Host Service Provider, illustrated on the right.



OAuth 1.0a Authentication Flow and Retrieval of Protected User Data

All requests sent from the guest service provider (aka *OAuth Consumer*, in this case the Distributed Social Networking system) are signed according to the OAuth specifications. Unsigned requests are rejected by the host service provider (aka *OAuth Provider*, in this case, Twitter) as unauthorized or invalid.

As can be seen, the authentication flow is burdensome and requires several connections between the consumer and the provider. On the positive side, however, this model is very secure and reliable.

Before OAuth was used by service providers, the only way to access protected user data hosted on another service provider was to ask for user credentials on the host service provider, and impersonate a user by sending HTTP requests to the UI of the host

service provider—just as the real user would do from within a web browser to access her account data. This model, however, comes with a bundle of disadvantages. First of all, the user is asked to provide their username and password of the host service provider to the guest service provider. Not all users are willing to share their credentials on multiple service providers with each other, essentially giving them full access to their personal data on the host service providers. Additionally, if the user changed their password on the host service provider, the guest service would need to ask the user for the updated password to be able to continue retrieving data from the host.

OAuth, in contrast, removes the need to ask for and store users' credentials on other services. Instead, through the authentication flow illustrated in the previous diagram, users complete the authorization process on the host service provider and, through a series of secure communications, the guest service provider is granted an access token which could be used to retrieve user data hosted on the host service without disclosing the credentials to the guest service provider. Additionally, OAuth 2.0 further simplifies the authentication flow by using the industry standard SSL connections instead of client side request encryption and signing, as well as removing steps one and four specified in the above diagram.

Regardless of the specifics of the OAuth authentication flow, once the access token for a particular user is granted by the host service to the guest service, it can be used to make API calls on behalf of the user and retrieve protected data. Though different in implementation, Facebook, Twitter, Linked in, and Flickr all adhere to the same concept of authentication and provide APIs and interfaces that have been used in this project to access user data. Implementation details specific to each of the service providers are provided in chapter five.

3.2.2. Data Extraction

Once authorized, the Distributed Social Networking system can start retrieving data from the four social networking service providers, Facebook, Twitter, Linked in, and Flickr. The aggregated data is used to carry out the declared mission of the project, which is providing a centralized online profile for the users, and providing social recommendations based on their areas of interest.

Each of the major service providers provides a unique API architecture and protocol. However, they all share HTTP as the communication protocol and provide access to their APIs through port 80, the standard web browsing port. Therefore, these APIs, though different from the Web Services, share the advantage of trouble free access through the firewalls, and utilize the full stack of HTTP facilities for network authorization, communication, and flow control. As such, the selected service providers offer RESTful Web Service interface to their APIs. The benefits of RESTful interfaces, in short, include reduced setup and operational overhead—by doing away with SOAP requirements such as a well defined WSDL document—flexible data format, and leaner, quicker validation and parsing processes.

Most of the service providers show flexibility in the data format of their return values. JSON and XML are the de facto standard data structures supported by Facebook, Twitter, Linked in, and Flickr. For simplicity and reduced overhead, XML responses are not accompanied with XSD references or specific namespace declarations. JSON is a much leaner data structure in nature, therefore more efficient in communication across the network, but lacks metadata and descriptive tags. In general, the expected data format returned by the service providers must be known to the service consumer for successful data extraction.

3.3. Data Retrieval Scheduling

The Service Interface and Scheduler component runs as a long living service within the Distributed Social Networking system. This core component utilizes a scheduler to refresh the data of all users at the desired interval. At the same time, this component exposes a method for instantaneous, on-demand data retrieval when requested by the UI component.

The Service Interface component is the data feed provider for the UI and the Data Mining components. Whenever new data is retrieved from external service providers, it is both persisted in a database, passed on to the Data Mining component, and is also sent to the UI if required.

To maintain the latest user information across all the Distributed Social Networking systems, a background scheduler is set, which triggers a set of methods on a specified interval. A scheduler is customizable and can be used with applications of any size and supports triggering any number of jobs. The following are a few scheduler solutions which we've approached each of which supports different levels of customizability.

The Java timer framework consists of standard scheduling classes Timer and TimerTask. Used by itself, it does not completely support daily scheduled tasks since as it does not support daylight savings time changes. However, the timer framework used in conjunction with a scheduling framework, creates schedulers which can run jobs at a certain time every day until the scheduler is canceled. This scheduling solution is best for simpler applications which do not require such flexible scheduling rules such as a single-run task, or a daily-run task for example an alarm clock application in which it sends an alert at the same time every day.

The scheduler framework consists of the Scheduler, SchedulerTask and SchedulerIterator classes. For each instance of the Scheduler, it owns an instance of the Timer which provides the actual scheduling. The Scheduler manages a set of single-run timer tasks which is combined into a SchedulerTask and is run at the times specified by the ScheduledIterator class.

The SchedulerTasks tasks enter the following states:

- VIRGIN: task has never been scheduled
- SCHEDULED: task has been scheduled, task can only be scheduled once
- CANCELLED: task has been cancelled

This scheduler framework does not utilize a thread-pool, therefore it is important to synchronize on the task's lock object else it may run an already cancelled task or double schedule a task (not supported). These limitations are reasons why this scheduling solution are best for more simpler applications. For complex scheduling requirements, a more flexible and configurable scheduling solution is recommended.

Quartz Scheduler is an open-source, highly configurable scheduler which used by thousands of users such as Cisco, Adobe, Apache and JBoss to name a few. Quartz can be run as a stand-alone program, within another application as well as be instantiated withing an application server or as a cluster of stand-alone programs.

Quartz is a small Java library which requires very little setup and configuration to get working. Similar to a cron job, Quartz supports even the simplest to the most complex schedules such as jobs starting on specific times, day(s), months, year with varying repetition rules. Quartz has the ability to participate in and manage Java Transaction API (JTA) transactions via JobStoreCMT during which the triggered methods are executed.

One feature that Quartz has, which not many schedulers have is support for job persistence, JDBCJobStore which is stores all 'non-volatile' jobs and triggers in a relational database via JDBC. Additionally, RAMJobStore stores all jobs and triggers in RAM which has the benefit of not requiring an external database.

There are many benefits of Quartz scheduling compared to the standard Timer and TimerTask classes.

- Quartz supports scheduling based on dates, times, time of day and varying repetitions whereas the Timer class only supports a set start-time and repeat interval scheduling.

- Quartz utilizes a thread-pool compared to the Timer class which supports only one thread per timer.

- Quartz has a persistence mechanism and the Timer class doesn't

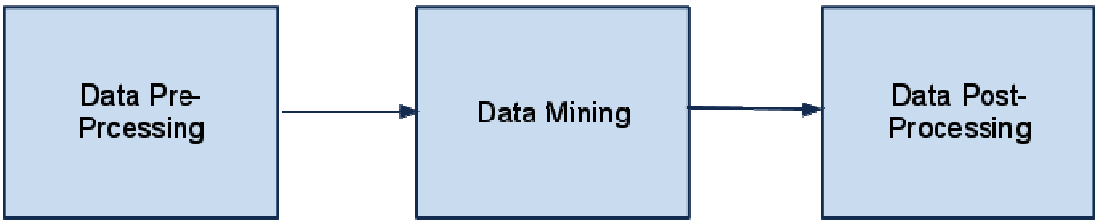
- Quartz keeps a history of job execution, can load job and trigger definitions from a file while the Timer class has no management schemes.

Given its customizable features, it is no surprise that Quartz has been adopted as a scheduler standard by many organizations.

There are many other scheduling frameworks available, the above described being the main frameworks we've explored. A much simpler scheduler solution is a java loop with a sleep delay. This is best for tasks that have no strict restrictions on when tasks are to be run or the frequency.

3.4. Data Mining Technologies

The Data Mining component of the Distributed Social Networking system is responsible for analyzing data fetched by the Scheduler and providing intelligence services to the UI. The overall goal of this component is to be able to discover patterns and similarities in user generated content. As the user generated content is mainly represented as text, the data mining component needs to support natural language processing mechanisms. A generic Data Mining process can be described as follows:



Data Mining Process

The Data Pre-Processing involves transforming incoming data from its original format into a form that is required by the particular data mining algorithm. The first step in the Data Pre-Processing is cleansing. Cleansing mainly includes punctuation and removal of stop words—words that are frequent but do not add to the semantics of the text such as prepositions, articles, etc. Moreover, the pre-processing step relies heavily on the natural language processing mechanisms such as text tokenization and stemming.

The Data Mining step of the process involves the implementation of algorithms and related data structures that are responsible for recognizing patterns in the incoming data. For the Distributed Social Networking system, the term frequency vectors and inverse indexes were used.

The Data Post-Processing step is responsible for generating aggregated reports such as the top N terms used across the entire data set, the top N terms used by a single user, the top N similar users based on their term vectors, etc. The described functionality should be accessible from a simple interface.

At the implementation stage of this project, the following frameworks were considered as basis for the Data Mining component:

- Apache Mahout (http://mahout.apache.org/)
- Apache Lucene (http://lucene.apache.org/)

Apache Mahout is an open source framework which goal is to provide implementation of various machine learning algorithms in an efficient and scalable fashion. The scalability of the framework is achieved through tight integration with the Apache Hadoop project, which provides an open source implementation of the map/reduce algorithm. Together, they provide an easy scalable solution that can process very large data sets efficiently while being deployed on commodity hardware.

The data mining aspect of the Mahout framework includes implementations of a variety of algorithms that generally can be divided into the following four categories:

- Collaborative Filtering
- Item-based Recommenders
- Clustering algorithms
- Classifiers

While offering a reach set of features that can be utilized in the Distributed Social Networking system, the Mahout framework is at the early development stage with the current version being 0.4. The API of the framework is not stable and with the core libraries undergoing heavy development, the potential for critical bugs is high.

Lucene is an open-source library that provides search and text analysis functionality. Lucene includes a stable API and mature implementation of the core information retrieval components. The current stable version (v. 3) offers a reach set of text analyzers, a high-performance term vector processor, and a near real-time full-text indexing and search.

In addition, Lucene is a very compact library with minimal footprint which allows it to be embedded into existing and future applications with little effort. Lucene has an extensive community support, with many modules contributed by third-party developers. Some of those modules were used in this project, including FastVectorHighlighter and WhiteSpaceFragmentsBuilder.

Above all, Lucene is a mature and well tested. It has been in active development since early 2000s and was successfully deployed in various projects by the industry leader such as Yahoo, Netflix, LinkedIn and SalesForce.

Overall, Lucene provided all the functionality required by the Distributed Social Networking system in a simple and lightweight package. The implementation is mature and well tested which makes it an ideal candidate for deployment into production. Lucene was used in this project as the implementation base of the Data Mining component.

3.5. Web Services for Distribution

One of the most important and challenging architectural requirements for the Distributed Social Networking system has been to design, implement, and deploy it in a distributed environment from the ground up, mimicking characteristics of a true enterprise software system. The motive behind this decision has primarily been to gain hands-on experience of how real enterprise systems span across multiple servers and scale horizontally.

The challenge, obviously, has been to create a distributed environment within the means and resources allocated to the project. Since the project consists of three distinguishable components—Service Interface and Scheduler, Data Mining, and User Interface—our approach was to create each of these components as a separate, standalone project deployed on a physically independent server.

3.5.1. Communication across Components

Distributed systems naturally need to be equipped with means for communication across servers in order to successfully orchestrate and carry out procedures that involve processes hosted by various components running on physically separate servers. The industry approaches for addressing this particular challenge are diverse and plenty. For instance, in a homogenous Java environment, JMS could be employed for communications across servers over the network. In a diverse environment, where processes created by multiple programming languages need to communicate with each other, other technologies and techniques such as CORBA and socket programming could be utilized. CORBA requires support from all participating languages, which are not very diverse due to a general decline in the technology's popularity. Socket programming, though extremely flexible and efficient, comes with its own set of intricacies, especially with regard to development and maintenance of the communication protocols. Furthermore, many of the communication and messaging technologies rely on open network communications through ports that might be normally blocked by firewalls along the route.

A viable alternative to aforementioned communication technologies is SOAP Web Services. Web services have been increasingly utilized in enterprise applications and are the new industry choice for widely portable and flexible service interfaces across a diverse host of platforms and programming technologies, languages, and frameworks.

By utilizing the TCP/IP and HTTP transport and communication protocols, Web Services utilize widely accepted and well established data transmission standards and infrastructures. However, the flexibility and robustness of web services come at the cost of complexity associated with WSDL documents, the inherent bulkiness and excessive overhead of XML documents—as the de facto default data structure passed between web services, the processing and time costs associated with validation of XML documents against XSD, DTD and other data/document definition documents, as well as issues with security and data encryption, to name a few. Altogether, where the shortcomings of web services can be afforded, they are suitable solutions for communication across servers and components of an enterprise software system.

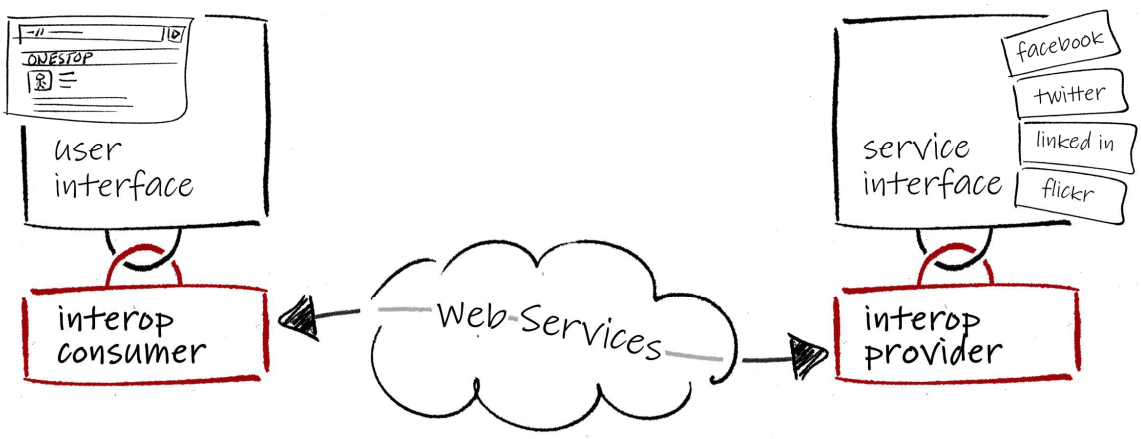
The Distributed Social Networking system consists of three major components: Service Interface and Scheduler, Data Mining, and the User Interface. The Service Interface and Scheduler, and the Data Mining components have been developed in Java. The User Interface component utilizes a hybrid of Java and .net environments, with web pages being hosted and processed on a .net enabled web server, and UI data aggregation services implemented in a Java backend.

3.5.2. The Web Service Interoperability Infrastructure

The bridge across all components of the system, which can be deployed on four physically separate servers, is a web service infrastructure wrapped in an extremely easy to use interop library developed in Java and in .net, and referenced by all components of the system. The interop library allows the referencing components to provide services and also consume services provided by other components.

The underlying communication technology utilized by the interop library is web services, as web services are deemed sufficient and suitable for the diverse environment of the Distributed Social Networking system. However, since the communication details have been concealed by the interop library, the current choice of web services could be substituted with any alternative distribution technology with absolutely no requirement for any code change in the referencing applications.

The following figure helps depicts how the provider and consumer classes of the interop library allow for communication across components through web services.



Interop provider and consumer classes conceal web service implementation and invocation details and enable components to communicate seamlessly

The consumer component, for instance the User Interface component in the above diagram, are unaware of the location of the service they consume and perform method calls on a local object which, through the interop library, are translated into web service requests, transmitted across the web service interface, processed by the provider components, the Services Interface component in this case, and receive a response object back. The interop library acts as a proxy between remote components and enables them to invoke remote methods the same way local methods are invoked. The interop library even propagates exceptions thrown by the provider component back to the consumer component, which can catch and handle them as desired.

As illustrated in the diagram above, a benefit of the abstraction layer created by the interop library is that the web services interface can be replaced by any other distributed communication technology, such as CORBA, socket programming, and common data and command bus techniques, without any change to the code and the business logic of components that use the interop library.

Since the components in the system provide a variety of features and functionalities, and in order to simplify change management processes and increase expansibility, the web services interface included in the interop library are unaware of the details of components that they provide interoperability services to. This goes against the common practice of web services development, which requires comprehensive definition of all methods and messages in a WSDL document before web services are deployed. Instead, a very lean WSDL document has been devised that supports only one method for calling remote methods on destination components. This web service method only needs to know the method name and the parameters that need to be passed to the remote component for valid invocation. The method name is specified as a string field, and the parameters are serialized from a Dictionary collection object into a JSON-like string. The

combination of the method name and the parameters is used along with reflection technologies, provided by Java and .net frameworks, to find and invoke the proper method on the remote components. The return values from the remote method are also serialized in the same JSON-like string and passed back to the consumer component, where they get deserialized into a Dictionary collection object.

As can be seen, the decoupling of the business logics among components and the interop library has resulted in a non-intrusive, highly maintainable and expansible distribution model which can inherently accept any degree of change to the business logic and communication technologies utilized in implementation of the system components and the component interfacing library.

3.6. User Interface Design

One of the main features of the Distributed Social Networking system is to provide a centralized profile for users where they can view the collective current state of their social networks, aggregated from Facebook, Twitter, Linked in, and Flickr. Distributed Social Networking does not intend to replace the other social networking services, but aims at creating a hybrid profile page where users can see an integrated list of updates posted on various social networking websites. The data is gathered by the Service Interface component—specifically by Service Bots—and is then passed on to, and stored in a database residing with, the UI component.

3.6.1. Lag-Free Data Access for UI

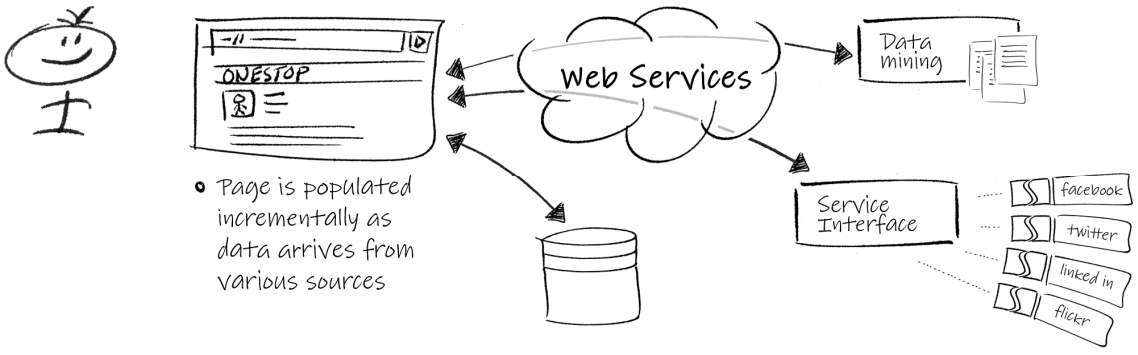
The UI component utilizes a MySQL database for fast OLTP type processes including user authentication and authorization, profile presentation, search operations, and assistance with account setup with the external service providers. The database serving the UI component needs to be fast, particularly for read operations, in order to assist with speedy user and profile lookup, and page rendering. Since the end users interface with the UI component, it is critical to demonstrate minimal response times and avoid any lag, if possible, for data retrieval. This will directly influence users' impression of the overall system performance and quality.

3.6.2. UI Design and User Experience

It is essential for the Distributed Social Networking system to sport a highly streamlined, intuitive, user-friendly and competitive UI design to gain any attention from the users of services such as Facebook, Twitter, Linked in, and Flickr. These major social networking service providers have, over time, accustomed users to highly perfected and professionally designed interfaces that fashion appealing graphics and layouts and rely on web 2.0 technologies and practices, including unobstructive ajax communications and simplified interactions. In such context, any attempt to attract users to any new social networking service on the internet—specifically the Distributed Social Networking project in this case—would require a comparably well designed user interface.

In order to produce an appealing user interface, pioneer front-end technologies such as HTML5, CSS3, Javascript and jQuery 1.4 have been utilized along with simple graphical elements. One of the challenges associated with the web interface design of this Distributed Social Networking system is to neatly combine and display data gathered from four service providers. Furthermore, the web pages need to include sections allocated to the output of the Data Mining component, which include a list of users with similar interests, a list of frequently used keywords by specific user, and by all users collectively. The design approach has stayed away from too many bells and whistles and has instead focused on practicality and simplicity of the user interface. By utilizing visual elements, dynamic positioning and animations, the limited real estate of the web pages has been carefully utilized to compose an easy to follow flow of information and interactions with the users.

Ajax has been extensively utilized to refrain from unnecessary page navigations and, instead, populate page data in an asynchronous and unobstructive manner. As a result, users are less distracted from the page views and also spend less time waiting for complete page reloads and redraws. In a distributed environment where data is aggregated from multiple sources and servers, including distributed internal components connected to each other through web services and external social networking service providers, wait times for data population can be significant, and failures can occur anywhere along the communication channels across the host of servers and components. As a result, it is essential to show data to the user incrementally as they are aggregated from different sources.



Incremental page population results in higher perceived quality of service

Due to its strength, robustness, flexibility, state-of-the art IDE, and rapid development cycle, ASP.net has been employed as the framework of choice for implementing the presentation tier of the User Interface component. The web pages inherit their base design and content layout from a master page that specifies and renders the basic template of all pages. This creates consistency among all pages pertaining to the Distributed Social Networking website—branded as “OneStop” for conciseness and simplicity.

3.7. Data Storage Options

To anticipate the trend of a rapid growth of user generated content that is evident all across the Web, the Distributed Social Networking system can take advantage of DHT-based file systems to store its users' content, along with the associated metadata, at a large scale.

The Distributed Hash Tables are similar to a regular hash table in a sense that each data record is stored as a key/value pair, where the 'value' represents the data itself and the 'key' is a hash function of that content. The difference from the regular hash table lies in the distributed underlying infrastructure of the DHT. In essence, DHT is a distributed network which main purpose is to efficiently store data.

The key features of a DHT are as follows:

- Decentralization: the nodes collectively form the system without any central coordination.
- Scalability: the system should function efficiently even with thousands or millions of nodes.
- Fault Tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

(http://en.wikipedia.org/wiki/Distributed_hash_table)

The most prominent implementations of the DHT-based distributed file systems include the Apache's Cassandra (http://cassandra.apache.org/) and LinkedIn's Project Voldemort (http://project-voldemort.com/). Both projects are open-source, at a fairly stable stage and with active developer community. Both projects can be utilized by the Distributed Social Networking system to provide large scale storage facilities for its data if such need arises.

Chapter 4. Project Design

The project design for the Distributed Social Networking project was devised after extensive research and prototyping, and examination of various alternative technologies as outlined in the previous chapter. A long and thorough thought process, hours of discussions, and creation of small, working prototype models of the software components described in earlier chapters resulted in a very smooth design and implementation phase. Pursuant to the guidelines of the project advisor, and according to the experience and knowledge garnered through the prototyping/proof of concept development phases, the following project design has been created and utilized for implementation of the system. It is noteworthy that the project design document has been kept lean and concise to simplify understanding and also remove complexity from the implementation phase.

4.1. User Interface

No matter how sophisticated a software system, end users only see and interact with the front-end user interface, which can singlehandedly determine the level of quality and reliability perceived by the users. Well designed user interfaces can attract users to the software system and make their experience with the system smooth and pleasant. Poorly designed interfaces, on the other hand, can easily bore or even annoy users and lure them away. There are several approaches to UI design for software systems and the fields of User Experience engineering and Anthropology provide scientific insight into principles of creating successful user interfaces.

User interface design consists of a number of important factors and elements that go hand in hand in affecting the overall user experience. Layout of text and graphics, color themes, font and size of text, utilization of audio and visual elements and animations, intelligent design of action workflows and navigation strategies, proper placement of information, and consistency among screens are among important factors that can significantly improve the quality of service as is perceived by system users.

4.1.1. UI Design Principles and Practices

More software systems and services are turning to web interfaces as their primary point of interaction with end users. Banks, news and publishing organizations, the hospitality, tourism, and entertainment industries, several branches of governmental organizations, and many other service providers are utilizing the web interface extensively and primarily as the point of contact with their customers. What pushes the web design practices and standards forward, however, are businesses and services that are online and web-based in nature. Email service providers, blogs, search engines, and virtual society portals are pioneers of new web UI design paradigms.

Facebook, with over half a billion users, was not the first social networking website introduced to web, but has certainly gained the largest amount of attention and interest from the online community. Its success is owed partly to a robust and open platform, and partly to an intuitive, easy to follow user interface that has employed web

2.0 standards and enhancements from its early days. A very appealing design, targeted carefully at the average web user, has contributed significantly to the popularity of this outstanding social networking service provider.

Twitter interfaces with its users primarily through the text messaging service provided by cell phone carriers. However, any function other than posting tweets and receiving those of other users, requires logging in to the website which, like Facebook, enjoys a sleek design and implements features according to web 2.0 practices and standards. Twitter has often been criticized for its less than perfect software architecture and recurring issues with regard to its scalability and reliability; however, a welcoming user interface design might have helped retain users who might have otherwise left the service due to its frequent technical issues.

Linked in and Flickr are probably some of the furthest lagging services with regard to incorporation of new UI design principles. They do seem to have hung onto a more traditional UI architecture, characterized by an excessive number of pages and frequent page navigations in order to perform simple tasks. However, both Flickr and Linked in, have recently made efforts to incorporate more elements that are elegant and charming to the eye, and to rethink and simplify user interaction procedures.

4.1.2. User Interface Design Considerations

The user interface design for the Distributed Social Networking project is a challenging task. First, because it needs to create an integrated interface where data aggregated from four other social networking service providers is viewed in a logical and easy to grasp fashion. Secondly, users of these four social networking service providers are accustomed to the design and interaction flow of the respective services and probably enjoy certain aspects of each other services. Therefore, they would expect the same or a similar quality of design and flow of actions from a service whose aim is to be a comprehensive combination of all other social networking service providers.

While the user interface of the Distributed Social Networking website cannot be a compilation of everything found on the other service providers it interfaces with, it does need to be elegant, appealing, and attractive to encourage users to use this website along with the other service providers. With this requirement in mind, the user interface has been designed with a minimalistic approach which, still, provides promised features through a sleek interface design and a streamlined user interaction architecture.

Web 2.0 design principles and practices have been employed, within the available means and resources, to create a user interface that meets the expectation of users of social networking services, such as Facebook, Twitter, Linked in, and Flickr. The design has been kept simple and practical and sports only what is required to use and experience the important features provided by the Distributed Social Networking system.

4.1.3. User Interface Sample Screen Shots

Though screen shots are incapable of expressing the full user experience with a dynamic, Ajax enabled web page, followings are a few screen shots of the overall look and feel of the Distributed Social Networking website—branded as “OneStop” throughout the web user interface.

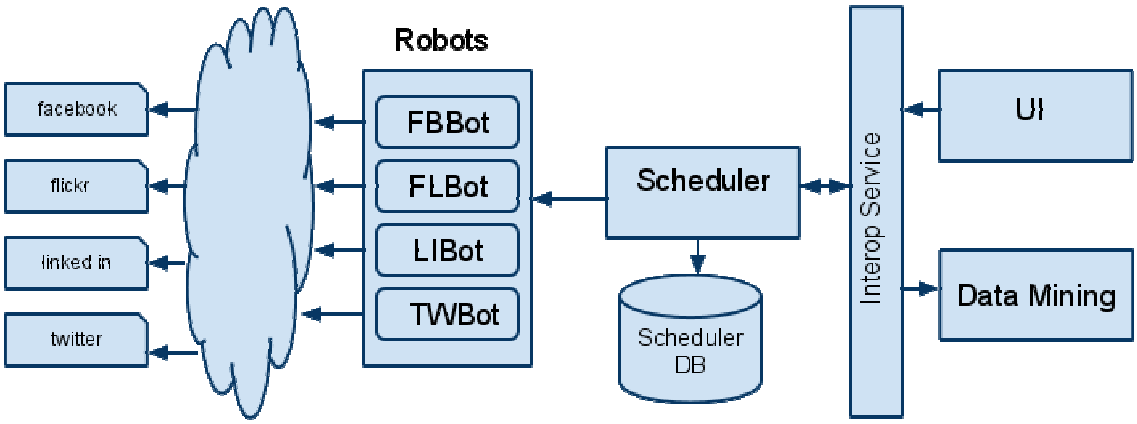
Integrated Sign in and Sign up page

Account Setup Page where users can connect their Facebook, Linked in, Twitter and Flickr accounts to their OneStop account

4.2. Scheduler Design

To retrieve the latest information from each service provider, a scheduler is set which runs at a continual specified interval. Each iteration of the scheduler runs through a set of methods, namely triggering the service provider robots to retrieve the latest information for each user propagate the data to the required systems. The scheduler allows for all systems in the Distributed Social Network to keep data in sync by propagating any data updates to both the UI and the Data Mining components.

We explored different scheduler solutions such as the Quartz Scheduler, Java Timer framework, simple loop with sleep delay (details for each which can be found in Chapter 3: Technology Description) and ultimately opted for a solution which we've designed specifically to the dynamic nature of the data we are processing: Java Timer framework with a Scheduler framework. The scheduler requirements of the Distributed Social Networking system are quite simple; a daily scheduler which triggers several robots and propagates the data to the Scheduler database, Data Mining and the UI components as can be seen in the below figure.



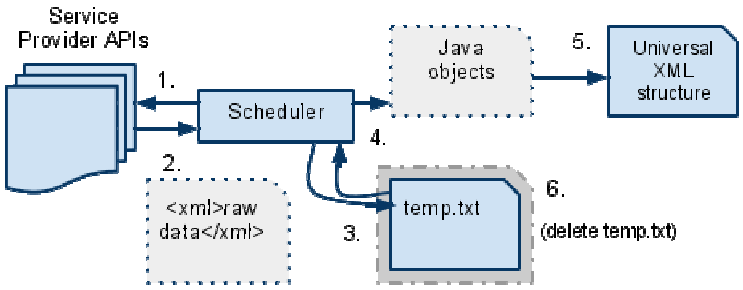
Interface Bots and Scheduler Components

4.2.1. Data Extraction Design

The initial stages of the Distributed Social Networking data extraction method was done by parsing and extracting data from an XML/JSON file on the filesystem. There was a layer data storage which was later called on for data retrieval:

1. Connect to the service provider API for latest user data
2. Returned response in JSON/XML (raw data) is temporarily stored on the local filesystem as a flat file
3. The flat file is read into a DOM/JSON object
4. Values are extracted from the DOM/JSON object and saved as separate Java objects

5. Store Java objects in a universal xml structure for all service providers per user
6. The flat file is deleted from the filesystem

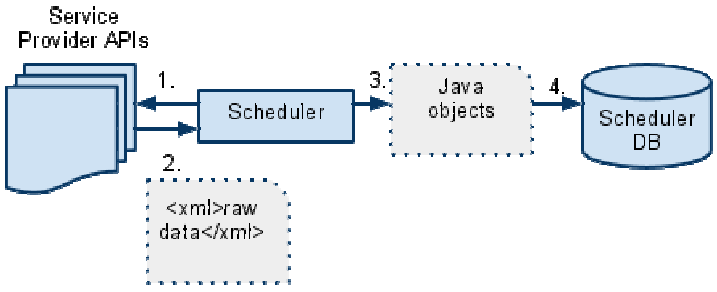


Data Extraction using Temporary Dump Files

Though this solution produced the desired objects, it consisted of multiple method calls which, if tested with a large number of users at the same time may cause noticeable overhead as well as potentially temporarily use critical filesystem space. Storing the extracted Java objects into a universal data structure (xml file) poses potential issues of a large file when a user's data grows large enough that it may cause overhead processing.

To efficiently parse and extract data while minimizing overhead, data is parsed and extracted as soon as the service provider HTTP response is received:

1. Connect to the service provider API for latest user data
2. Returned response in JSON/XML is saved as a JSON or XML object respectively
3. Values are extracted via a parser (XPath, JSON library) and saved as Java objects
4. Store values in the database as needed



Data Extraction and Parsing using XPath and JSON Libraries

4.2.2. Data Format Handling

JSON and are XML are two of the commonly used and standard data formats in data-interchange. Both maintain a structured data format, yet have slight differences which depending on the purpose, may have preference to use one over the other.

JSON is a text format, derived from JavaScript and is built on both a collection of name-value pairs and an unordered list of values. JSON is ideal for serializing a data structure that is not text-heavy. Its ease of writing, reading and parsing of the data structure allows JSON to be supported by virtually all programming languages. JSON supports the following data type values: string, number, object, array, true/false and null. A JSON object begins with a left brace '{' and ends with a right brace '}', each name an value pairing, which are separate by a colon ':', are listed with comma ',' delimited. For example:

```
{
  "data": [
     {
        "name": "APPLE,
        "id": "113054308732506"
     },
     {
        "name": "Starbucks",
        "id": "22092443056"
     },
  ]
}
```

Data is extracted by using expressions to retrieve a particular key-value pair, each of which can be defined as an object for later utilization. There is also a org.json library which can quickly parse JSON structures and extract values to Java objects. An exmple of creating a JSON object:

```
testString = new JSONObject().put("Message",
                    "Greetings!").toString();
```

which creates the string:

```
{"Message": "Greetings!"}
```

XML is a text format, derived from SGML and was originally designed to handle the demands of large-scale electronic data transfer. XML does not do anything other than store, maintain the structure and transfer data and is commonly used in compliment to HTML. XML is W3C recommended so it is the most commonly used for all types of applications.

XML format consists of tags, both pre-defined and user-defined, used to define the data structure. The following is a simple example of a user's data of service provider accounts:

```
<user>
      <profile>
            <services>
                  <serviceProvider spName="flickr">
                        <id>dsn_cldulay0504</id>
                        <realName>Crystal D</realName>
                  </serviceProvider>
            </services>
      </profile>
</user>
```

There are several solutions in parsing the XML data such as XPath, converting XML into an DOM object and JQuery to name a few. The quickest approach to parse XML data is with XPath, the data can be parsed and extraction without the overhead incurred when converting to objects as in the DOM method. Details of these are described in section 'Data Parsing and Extraction'.

4.2.3. Data Parsing and Extraction

The most common data parsing and extraction tools available are XPath and the JSON library, the latter of which applies only to JSON data. Each of these allows for the extraction of specific data values on the HTTP response as well as on a flat file, making any of these tools an ideal choice for stand-alone applications or distributed large-scale applications.

XPath Expressions

Using path expressions, XPath navigates through the elements and attirbutes of an XML or JSON data structure to retrieve the data values. XPath contains a library of over 100 standard built-in functions and is recommended by W3C. XPath provides wide range of expressions to extraction data as can been seen in the following tables of sample expressions.

Expression	Description
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

XPath Selectors and Useful Expressions

XPath uses predicates to retrieve nodes contain specific values, denoted with square brackets as in the example path expressions below:

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** IE5 and later has implemented that [0] should be the first node, but according to the W3C standard it should have been [1]!!
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

XPath Expressions using Predicates

In addition to retrieving specific nodes, XPath can also retrieve unknown nodes by using wildcards, denoted by an asterick '*', and also retrieve several paths using the pipe character '|', ie. '//title | //price' to select all title and price elements within the document.

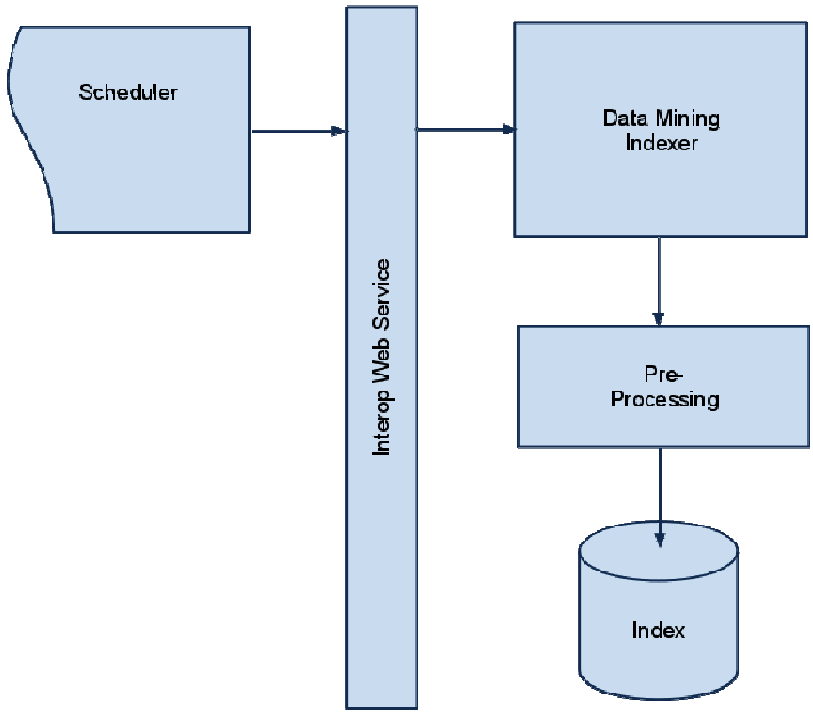
JSON Library

The JSON library parses JSON data structures, namely the *JSONParser* class which creates an object *JSONArray* or *JSONObject* whose data values are extracted by specifying the node name. *JSONArray*, is an ordered squence of values, with a consturctor which can convert JSON text into a Java object. JSONObject, an unordered collection of name-value pairs, can get data by a simple *get*()* method based on any of the supported data values: boolean, double, int, long, array, object, string.

4.3. Data Mining Component Design

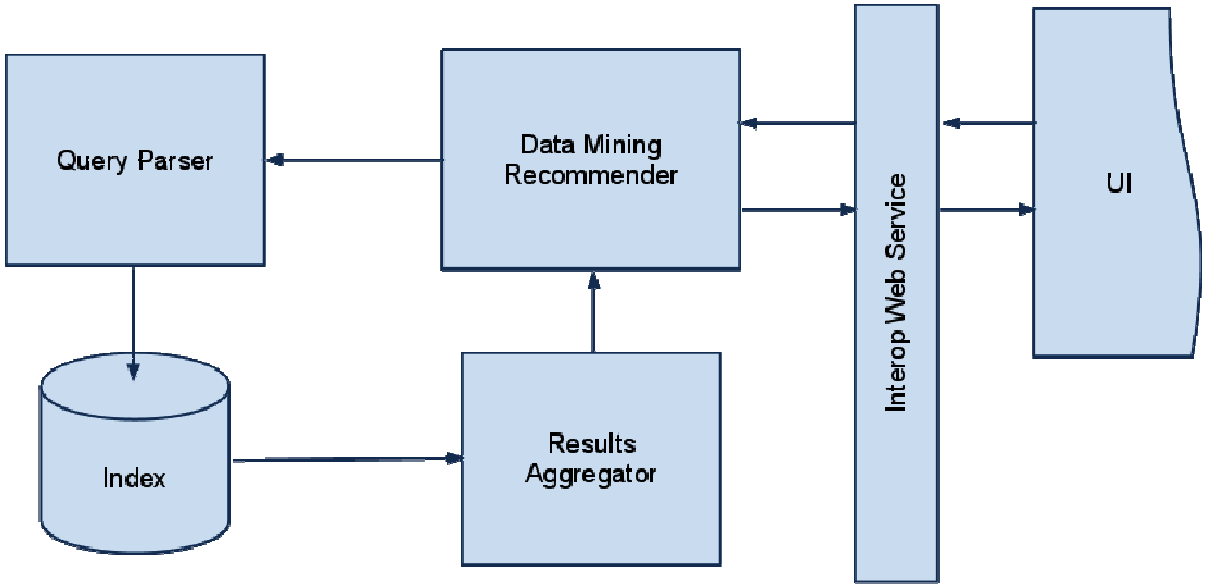
The Data Mining component of the Distributed Social Networking system plays an important role in the overall flow of data through the application. It must be able to receive updates from the Scheduler component asynchronously while handling user queries coming from the UI. To be able to handle this functionality efficiently the Data Mining component is divided into three logical parts:

- Indexer – receives updates from the Scheduler
- Recommender – handles user queries from the UI
- Persistent Index – connection point between the Indexer and the Recommender



Data Mining Indexer

Indexer sub-component is implemented as the Interop Web Service Provider that is able to receive asynchronous messages from the scheduler, extract the user specific information, pre-process it and update the local persistent index. To prevent concurrency issues, the Indexer needs to obtain a lock (implemented as a lock file in the index directory) before actually updating the index file.

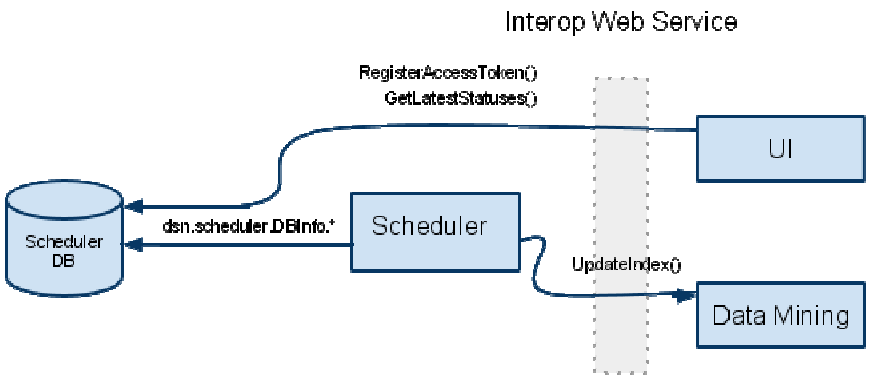


Data Mining Recommender

Recommender sub-component is implemented as the Interop Web Service Provider that handles requests coming from the UI component synchronously. It is able to distinguish between different types of requests (e.g., search, similar users, top terms) and query the local index appropriately. To improve performance of the concurrent requests, all Recommender's operations on the local index are read-only, since the Lucene library allows for the read-only operations to be performed without acquiring the index lock.

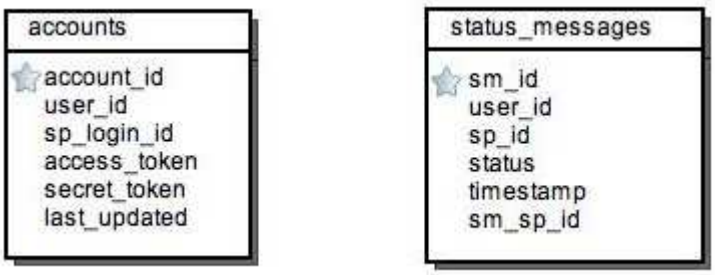
4.4. Scheduler Database Design

The scheduler database is used primarily to store user accounts and user status messages. The user accounts table is populated with data passed from the UI, essentially the data on the scheduler database is in sync with the UI component database. The user account information in the scheduler database is used by the scheduler to get the list of all user accounts for which the robots will be triggered. Both the UI and the Data Mining components access the scheduler database via the Interop Web Service.



Scheduler Database Queries

The above diagram shows the calls supported on the scheduler DB. Most of the calls originate from the Scheduler as there is one call per user account for each service provider, thus denoted by 'dsn.scheduler.DBInfo.*'. The UI is the only other component that makes calls to the scheduler database via *RegisterAccessToken()* and *GetLatestStatuses()*. The Data Mining component is indirectly updated with the scheduler database data when the scheduler runs. The following is the scheduler database schema:



Scheduler Database Schema

Chapter 5. Project Implementation

The implementation of the Distributed Social Networking project was made easy thanks to a solid architecture and a thorough yet simple system design, as discussed in earlier chapters. Following is a brief overview of implementation details of the major system components.

5.1. Scheduler Services

The scheduler implemented for our project is a combination of the java timer framework and a scheduler framework. We opted for this solution since our immediate need was simple; to run our scheduler at the same time daily during which a set of service provider robots are triggered to retrieve data. There are two types of calls supported by the scheduler:

- single-user *passDataThisUser()*: called when a user who is logged into their account and requests to retrieve their latest data from each service provider
- all-users *passDataAllUsers()*: the daily scheduler which runs daily, retrieving latest user data

The daily scheduler time is set in *passDataAllUsers*() in the form of the exact hour, date and second (HH:mm:ss) when the scheduler will run each day. When the scheduler runs, it triggers each service provider (Facebook, Flickr, Linkedin, Twitter) robot which checks on the service provider side for new messages that have not yet been persisted to the scheduler database by checking the unique value *sm_id,* the service provider assigned status message Id. This ensures that no duplicate messages exist on the system.

The following are the supported tasks for a single-user requested scheduler *passDataThisUser()*:

- retrieve latest user data from each service provider and persist in the database

The following are the tasks triggered during each run of the daily scheduler *passDataAllUsers()*:

- retrieve all user accounts from the database
- for each user account, trigger the respective service provider robot which retrieves latest updates and persists in the database
- pushes latest updates to the DataMining system
- trigger the updateIndexer task on the Data Mining system

For the purpose of the project demo and project exposition where we need to run the complete loop from user registration to displaying user data and mined data in the UI in a short period of time, we implemented a simple loop with a sleep delay,

runScheduler() to trigger the necessary tasks and robots. This temporary solution allowed for us to demo the scheduled tasks in a shorter period of time, say every 5 minutes rather than the actual once a day schedule. This way, our colleagues are able to see the scheduler in action within 5 minutes which they would not be able to do otherwise.

5.2. Service Interface Robots

There exists one robot per service provider: FacebookBot, FlickrBot, LinkedinBot and TwitterBot. The separate robots allow for current data retrieval which minimizes overhead and impact to the system in case of any service provider specific performance issues. Each robot is specific to that service provider and can support instant raw-data parsing as well as data retrieval from JSON and XML files. To reduce overhead, the instant data parsing method is used. When the scheduler triggers each service provider robot for each user, it runs the *runAll()* method which executes the following set of tasks:

getMessages():

Using the user's service provider token and secret (as required by the service provider), the robot connects to the service provider API for which to obtain the current messages. User token and secret are stored in the scheduler database so they must be retrieved prior to being passed to *getMessages()*. Each service provider has different requirements in accessing data via their APIs.

- Facebook: requires user access token which is appended to the API URL. For example: https://graph.facebook.com/me/statuses?access_token=<user-access-token>
- Flickr: similar to Facebook, for example: http://api.flickr.com/services/rest/&method=flickr.auth.checkToken
- Linkedin: requires user access token and secret which are passed along with the call to the Linkedin updates API: http://api.linkedin.com/v1/people/~/network/updates?scope=self&type=SHAR
- Twitter: requires user access token and secret which are passed along with the call to the Twitter statuses API: http://api.twitter.com/1/statuses/user_timeline.json

The response from the service provider API call is in either a JSON or XML format; Facebook and Twitter return data in JSON while Twitter and Flickr return data in XML. The returned data is parsed so only the *message, message_id* and *timestamp* are

returned to *getMessages().* Details of how the data is parsed are explained in the next section 'Data Extraction' section. The parsed data is then passed to *saveMessagesToDB()* to persist in the scheduler database;

saveMessagesToDB():

After the requested data is received from the service provider, a JDBC connection is made to the dsndb scheduler database and the returned *message*, *message_id* and *timestamp* data is inserted into the status_messages table.

Upon completion of all robots for all users, the scheduler runs *UpdateIndex()* which passes all new status messages since the last scheduled run to the Data Mining component.

5.3. Data Extraction and Parsing

The Distributed Social Network retrieves user information from the service providers via the APIs. There are many different APIs available for each service provider, and with the user's token, virtually almost all user information (non personal) can be retrieved. Without authentication, basic information available via the APIs are usernames, user's First and Last names to name a few. With authentication, more complete information are available such as status messages, user image, list of friends, and other service specific information. For our purpose, we use a more general API from which we are able to retrieve the needed information for populating the database with basic user information along with status messages.

Each service provider supports response types JSON or XML and in some cases both. Twitter, for instance, be default returns XML, but provides the option to convert data to JSON as what we opted. Our implementation handles the the following HTTP response types for each service provider listed in the following table. Also listed are the base API URLs used in requesting users information, the specific API type with the URL is bolded.

Service Provider	API URL	Response Type
Facebook	https://graph.facebook.com/me/**statuses**?access_token=<user-token>	JSON
Flickr	http://api.flickr.com/services/rest/&method=**flickr.auth.checkToken**	XML
Linkedin	http://api.linkedin.com/v1/people/~/network/**updates**?scope=self&type=SHAR	XML
Twitter	http://api.twitter.com/1/**statuses**/user_timeline.json	JSON (XML by default)

Each connection to the service provider is made with the call to *dsn.bots.<service_provider>.getMessages().* For JSON response types, as for Facebook and Twitter, the response is parsed as a *JSONParser* object then saved as a *JSONObject* or *JSONArray* object using the JSON libraries. The specific data objects can be extracted by using a *get()* on the specific data field. The Facebook and Twitter implementation differ slightly. As an example, a code snippet of the Facebook *getMessages()* implementation:

```
String resp = Utils
        .getUrl("https://graph.facebook.com/me/statuses?access_token="+token);
JSONParser jp = new JSONParser();
JSONObject jo = (JSONObject) jp.parse(resp);
JSONArray ja = (JSONArray) jo.get("data");

int size = ja.size();
String[][] messages = new String[3][size];

for (int i = 0; i < size; i++) {
    jo = (JSONObject) ja.get(i);
    messages[0][i] = jo.get("id").toString();
    messages[1][i] = CalendarHelper.formatTimestamp(jo.get("updated_time").toString());
    messages[2][i] = jo.get("message").toString();
}
```

Facebook getMessage() Implementation

If the response type is in XML, as for Flickr and Linkedin, the response is saved as a DOM *Document* object, which then can be extracted of its individual node elements and saved as *NodeList* objects. We initially must understand the response structure so we can specificy the exact fields to extract. The following is a code snippet of the Linkedin *getMessages()* implementation:

```
String resp = Utils.getUrl(
        "http://api.linkedin.com/v1/people/~/network/updates?scope=self&type=SHAR",
        ServiceProvider.LINKEDIN, token, secret);

Document d = null;
try
{
    d = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(
            new InputSource(new StringReader(resp)));
} catch (Exception ex) {}

NodeList nlTimestamp = d.getElementsByTagName("timestamp");
NodeList nlUpdateKey = d.getElementsByTagName("update-key");
NodeList nlStatus = d.getElementsByTagName("current-status");

String[][] messages = new String[3][nlTimestamp.getLength()];

for (int i = 0; i < nlTimestamp.getLength(); i++)
{
    messages[0][i] = nlUpdateKey.item(i).getFirstChild().getNodeValue();
    messages[1][i] = nlTimestamp.item(i).getFirstChild().getNodeValue();
    messages[2][i] = nlStatus.item(i).getFirstChild().getNodeValue();
}
```

Linked In getMessage() Implementation

Note that once the JSON and DOM objects are saved, they are put into the correct return structure *messages[][]* to be sent to the scheduler and UI systems. The code snippet examples are small subset of the many data elements that can be extracted from the service provider API responses.

5.4. Data Mining Implementation

The implementation of the Data Mining component relies heavily on the core Lucene classes as well as some third-party modules from the Lucene's 'contib' package, including:

- Lucene's IndexWriter – used to update the local index
- Lucene's IndexReader – used to read from the local index
- Lucene's StandardAnalyzer – used to perform text tokenizing and stemming
- Lucene Contrib's FastVectorHighlighter – to retrieve text snippets
- Lucene Contrib's WhitespaceFragmentsBuilder – to generate whitespace-bounded text fragments

Custom components include:

- DsnIndexer – encapsulates core indexing functionality
- DsnSearcher – encapsulates core searching and recommendation functionality
- DsnAnalyzer – allows for chaining of text analyzers
- Customization of WhitespaceFragmentsBuilder – to generate N word-sized text fragments
- Integration with the Interop Web Service – to allows communication with other components

Following is the detailed description of the main classes.

DsnIndexer

DsnIndexer class is responsible for updating the local index with user generated content. Before saved into the index, however, the content needs to be processed by the analyzer (see section on the DsnAnalyzer for details). Once processed, the resulting data is saved into the local index using the Lucene's IndexWriter object. The local index maintains a separate record for each user that is identified by the system wide user ID. The user's record contains the original text representation of the user's content, plus, a term vector created by the DsnAnalyzer for that content. All subsequent updates created by the user are appended to the user's record in the index and the term vector is recalculated.

DsnSearcher

DsnSearcher class implements the two main features of the Data Mining component. First, it provides the near-real-time search functionality across all user generated content that has been gathered and indexed by the system. DsnSearcher relies on the Lucene's internal querying and scoring mechanism to rank the results. The similarity formula that is used by Lucene is as follows:

$$\sum_{t \text{ in } q} (tf(t \text{ in } d) \times idf(t)^2 \times boost(t.field \text{ in } d) \times lengthNorm(t.field \text{ in } d)) \times coord(q,d) \times queryNorm(q)$$

Lucene's Similarity Formula (Manning)

`tf(t in d)`	Term frequency factor for the term (`t`) in the document (`d`)—how many times the term `t` occurs in the document.
`idf(t)`	Inverse document frequency of the term: a measure of how "unique" the term is. Very common terms have a low `idf`; very rare terms have a high `idf`.
`boost(t.field in d)`	Field and document boost, as set during indexing (see section 2.5). You may use this to statically boost certain fields and certain documents over others.
`lengthNorm(t.field in d)`	Normalization value of a field, given the number of terms within the field. This value is computed during indexing and stored in the index norms. Shorter fields (fewer tokens) get a bigger boost from this factor.
`coord(q, d)`	Coordination factor, based on the number of query terms the document contains. The coordination factor gives an AND-like boost to documents that contain more of the search terms than other documents.
`queryNorm(q)`	Normalization value for a query, given the sum of the squared weights of each of the query terms.

Methods provided by Lucene for Data Mining
(Lucene in Action, Manning)

The next feature provided by the DsnSearcher class is finding similar users. The users are compared according to the similarities of their term vectors (the actual formula is described in the subsection above). Each user is ranked according to his or her similarity score and the users with the top scores are returned.

A late addition to the DsnSearcher class was the functionality that allows to retrieve most frequent terms across the entire index and for each particular user. The implementation involves retrieving a term vector for an individual user or for all users in the index and determining the top N terms based on their frequency. For performance purposes, the processing of term vectors is implemented using a priority queue data structure and the frequency threshold (refer to the source code for details).

5.5. Scheduler Database Implementation

As described in Chapter 4: Project Design, the scheduler database manages the user accounts and status messages. The scheduler database, MySql, consists of two tables that are independent of each other. The primary keys are account_id and sm_id for the accounts and status_messages tables respectively, which are just auto incremented values.

The following are the table details for the scheduler database.

dsndb.accounts

```
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| account_id   | int(11)      | NO   | PRI | NULL    | auto_increment |
| user_id      | int(11)      | NO   | MUL | NULL    |                |
| sp_id        | varchar(2)   | NO   |     | NULL    |                |
| sp_login_id  | varchar(50)  | NO   |     | NULL    |                |
| access_token | varchar(200) | YES  |     | NULL    |                |
| secret_token | varchar(200) | YES  |     | NULL    |                |
| last_updated | datetime     | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
```

Column	Description
account_id	integer: account Id
user_id	integer: user's Id
sp_id	string: service provider Id (fb, fl, li, tw)
sp_login_id	string: user's login Id with the service provider
access_token	string: user's access token
secret_token	string: users' secret token
last_updated	datetime: timestamp when the user's data was last updated

dsndb.status_messages

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| sm_id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| user_id   | int(11)      | NO   |     | NULL    |                |
| sp_id     | varchar(2)   | NO   | MUL | NULL    |                |
| status    | varchar(500) | NO   |     | NULL    |                |
| timestamp | datetime     | YES  |     | NULL    |                |
| sm_sp_id  | varchar(100) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
```

The status_messages table has a unique index consisting of the *sp_id* and *sm_sp_id* values, this ensures that only unique status messages are stored in the table. A brief description of the status_messages table columns:

Column	Description
sm_id	integer: status message Id, database assigned
user_id	integer: user Id of the user
sp_id	string: service provider Id (fb, fl, li, tw)
status	string: status message
timestamp	datetime: time stamp of when the status update was made
sm_sp_id	string: unique string for status messages assigned by service provider

Chapter 6. Performance and Benchmarks

Considering that the Distributed Social Network manages its data from several service providers, there is some level of dependency on these separate services. If any one of the services go down or encounter any performance issues, the Distributed Social Network can still display users information and perform the Data Mining indexing and analysis, but only on the current data in the database. Any new data, as would normally be retrieved by the scheduler, will not be propagated to the Data Mining, scheduler or UI until the next successful user data refresh or scheduler run. Testing experience has encountered performance issues with Twitter availability more common than any other service provider. Fortunately the UI, scheduler and Data Mining systems are set up such that performance and data impact is minimal

Since access from the Distributed Social Network to the separate service providers requires an API key, there are limitations on the number of request calls that can be made within a certain period of time. The restrictions and time frame vary per service provider. We have hit these thresholds during heavier testing cycles which had temporarily disabled our systems from having the latest data, however still able to display and analyze the existing data. For more established and commercial use, these service providers enter deals with external entities to increase or completely remove the hit limit imposed on general applications. The Distributed Social Networking system has the potential, if it were to be turned into a commercial project, to push negotiations with its external service providers to allow for increased number of requests per unit of time.

The distributed nature of this project gives sense to the issue of performance when the number of users and amount of load on the service increases. The distribution itself is a remedy to potential resource limits encountered by deployment of a large project on one server machine only. Since the Distributed Social Networking system has been designed and implemented with distribution and scalability in mind, it is capable of easily scaling out in response to increased demand.

However, the web services, as the current underlying technical choice for communication among system components, while flexible and robust, may impose barriers and unnecessary overhead in case of increased activity on the system. Web services are criticized for their bulky messaging system and the extra, possibly unnecessary measures and processes. This has not been an issue in the development phase of this project, and web services are certainly the most sound choice for this system in its initial deployment and usage stages. However, should web service prove inefficient or burdensome on the overall performance of the system, they can be easily replaced with alternatives such as message passing through socket communications. Such substitute technology, while more flexible and more efficient, has its own shortcomings as well.

In short, the performance of the Distributed Social Networking system in our tests has been impressive and satisfactory. There are areas that could be optimized or substituted for even higher performance measures.

Chapter 7. Deployment, Operations, Maintenance

Following project dependencies and deployment procedures need to be met in order to deploy and run the Distributed Social Networking project.

7.1. Project Dependencies

The Distributed Social Networking system utilizes a few libraries that are referenced by the code and need to be present at compile time. These libraries provide functionality with regard to MySQL database connections and queries, Lucene for data mining procedures, and some other utilities, such as Simple-JSON which provide functionality for parsing and data extraction operations. The following libraries need to be downloaded and included in the classpath variable for successful compilation.

The Data Mining component relies heavily on the Lucene v.3 framework and a few third-party Java libraries and modules. Fortunately, Lucene is a fairly self sufficient software and does not require any additional dependencies. The complete list of the required libraries for the Data Mining component to function properly is as follows:

- MySQL Connector/Net
  Interfacing with MySQL database within the OneStop web application

- mysql-connector-java-5.1.13-bin.jar
  Interfacing with MySQL database within Java projects

- commons-codec-1.3.jar
- commons-discovery-0.2.jar
- commons-httpclient-3.1.jar
  Libraries providing Web Service connectivity and interoperability

- json_simple-1.1.jar
  Working with JSON object representations

- signpost-core-1.2.1.1.jar
- signpost-commonshttp4-1.2.1.1.jar
  Signpost is used for simplifying OAuth connectivity and interfacing

- lucene-analyzers-3.0.2.jar
- lucene-core-3.0.2.jar
- lucene-fast-vector-highlighter-3.0.2.jar
  Lucene is used for data mining operations

7.2. Project Deployment

Distributed Social Networking is implemented in form of five projects that need to be compiled and executed simultaneously for all system features to work properly. These projects are:

- dsn-scheduler (Java project)
  providing scheduling and service interfacing mechanisms

- dsn-datamining (Java project)
  providing artificial intelligence mechanisms and performs data mining operations

- dsn-ui (Java project)
  providing services to the User Interface project

- dsn-commons (Java project)
  providing web service interoperability features to all aforementioned projects

- OneStop (Asp.net web project)
  providing the end user interface in form of a website, communicating with all four aforementioned projects

The OneStop Asp.net web project needs to be deployed to a Windows Server machine utilizing the Internet Information Services (IIS) web server application. This web server needs to have .net framework 4.0 installed for the OneStop website to compile and run. Additionally, since OneStop interfaces with a MySQL database, an instance of MySQL database server needs to be installed and configured on the Windows server machine. Additionally, the MySQL Connector/Net library needs to be downloaded and installed on the Windows server to allow for interfacing of the Asp.net web application with the MySQL database. MySQL Connection/Net library is produced and maintained by MySQL and can be found at the official MySQL website.

Once the Windows Server environment is prepared, deploy the code of the OneStop website from the accompanying CD into the wwwroot folder of the default website as specified in your IIS settings. Asp.net enjoys hot compilation and does not require any manual trigger to start working. As soon as the files are copied to the wwwroot folder, the web application is compiled and started. The website will be available for viewing at http://localhost/ however, before viewing the website you need to install and run the other projects as well.

Each of the Java projects including dsn-scheduler, dsn-datamining, and dsn-ui need to be compiled and executed on a separate server. The file interop.config needs to be modified to correctly reference the correct IP addresses of all server machines that are running each of the projects. Please also note that these three projects reference the fourth Java project, dsn-commons, for their interoperability operations.

Each of the Java projects is self sufficient and contains all required libraries included in the lib folder found at the root of each project folder. There are two ways to compile and run each project. You could either utilize the “ant” build file and execute an “ant all” command, or alternatively, open each of these three projects in Eclipse. Eclipse configuration files included with the projects are self sufficient and allow for immediate resolution of dependencies and libraries and seamless execution of the projects.

The accompanying CD also includes DB Schema creation scripts in form of two .sql files. The database schema found in the OneStop folder needs to be executed against the MySQL database instance installed on the Windows server machine. The schema file included within the dsn-scheduler project needs to executed against the MySQL database instance installed on the machine that hosts the dsn-scheduler project.

Once all projects have been configured and deployed to their respective server machines, they can be started which causes their web service interfaces to publish their respective services on the specified end points and ports. The following screen should be displayed by all projects:

```
2010-12-01 11:43:28 INFO: Initializing the web service provider
2010-12-01 11:43:28 INFO: Loading INTEROP configuration
2010-12-01 11:43:28 DEBUG: GeneralRemoteMethod instantiated with interop handler
                           dsn.ui.UIServiceHandler

Dec 1, 2010 11:43:29 AM com.sun.xml.internal.ws.model.RuntimeModeler getRequestWrapperClass
INFO: Dynamically creating request wrapper Class
dsn.commons.interop.service.provider.jaxws.InvokeRemoteMethod
Dec 1, 2010 11:43:29 AM com.sun.xml.internal.ws.model.RuntimeModeler getResponseWrapperClass
INFO: Dynamically creating response wrapper bean Class
dsn.commons.interop.service.provider.jaxws.InvokeRemoteMethodResponse

2010-12-01 11:43:29 INFO: Web service successfully deployed at http://192.168.79.1:8888/dsn-ui
                          Press Ctrl+C to stop the web service
```

At this point, the web interface could be started by opening a web browser and navigating to the address http://localhost/ on the machine that hosts the Asp.net OneStop web application. Please make sure that all servers are connected to the same network, have been assigned valid IP addresses, have successfully authenticated through the firewalls, can respond to ping echo requests from each other.

Once the application is set up and all servers are running, you can invoke operations through the web interface and notice all servers working together to process requests initiated by the calls from the web interface. The servers print debugging data on their consoles and continue working until they are manually stopped by the system administrator.

Chapter 8. Summary, Conclusions, and Recommendations

Due to timing constraints, there were many features which we did not have the opportunity to implement, however this allows for plenty of room for additional feature enhancements. Currently the Distributed Social Network supports four of the top social networking sites, in later releases there can be support for more social networking sites such as MySpace, Bebo, Flixster to name a few.

The system design is such that it can easily support new service providers since this would not involve minimal, if any core functionality changes. Another feature enhancement is having additional data extraction which will allow the user to view and compile more of their data on the Distributed Social Networking site. Information such as groups belonged to, list of friends, and photos are all data that the user will have access to on the Distribute Social Networking site.

With the amount of data coming from various resources, there are potentials for data analysis and metric graphs to analyze overall users usage based on selected factors such as top topics or user location. There is definitely a lot of opportunity to enhance the user's experience to truly make this their "OneStop" Distribute Social Networking site.

This project has been designed and implemented merely as an academic endeavor in the emerging direction of service integration and interoperability among a diverse host of online service providers. The values created by this system for its users include simplicity, unity in design, increased potentials for virtual social encounters, and decreased amount of time spent browsing through multiple websites for performing basic social networking tasks. For the service provider, the value lies within the distributed and diverse mass of data gathered from external service providers. As demonstrated in this project, this data can be mined for the purpose of providing advanced and new features to the users of the system. Internet based businesses rely heavily on their user data for directing advertisement and turning the service platform to a profitable online product. Although this project throughout its conception and creation phases has not had any commercial planning associated with it, it does provide such platform for commercial purposes.

The project work has been a rewarding experience for the team members, as it has enabled us to research some of the most recent state of the art software technologies, techniques, practices, challenges, and gain a firsthand insight into the future of distributed software systems. This project has engaged the authors in lengthy and highly dynamic discussions for conception and development of its concepts and goals, as well as its tangible and practical ways of implementation and presentation. We are fortunate to have had the chance to work on such new concept and receive unwavering support from our advisors and instructors. We certainly recommend expansion upon this work by other students who are interested in the direction at which web applications are traveling and would be happy to provide any further information or assistance than presented in this document.

Glossary

API	Application Programming Interface
CSS	Cascading Style Sheets
DHT	Distributed Hash Table
DOM	Document Object Model
DSN	Distributed Social Networking
HTTP	Hyper Text Transfer Protocol
Interop	Interoperability—Ability of components of the system to communicate
OAuth	Open Authentication Protocol
OneStop	Name associated with the web user interface for the DSN project
JDBC	Java Database Connectivity
JSON	Javascript Object Notation
Lucene	Data Mining Library
SP	Service Provider—Referring to Facebook, Linked In, Twitter, or Flickr
UI	User Interface
WSDL	Web Service Definition Language
XPath	XML Document Parsing and Processing Library
XML	Extensible Markup Language
XSD	XML Schema Definition

References

DeCandia et al. Dynamo: Amazon's Highly Available Key-Value Store. (2007). Retrieved March 12, 2010 from http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

Dougherty Heather. Experian Hitwise. Facebook Reaches Top Ranking in US. (2010, March 15). Retrieved March 21, 2010 from http://weblogs.hitwise.com/heather-dougherty/2010/03/facebook_reaches_top_ranking_i.html

Dubost, Karl. W3C Oneliners. (2009, October 21). Retrieved March 10, 2010 from http://www.w3.org/2005/Incubator/socialweb/wiki/OneLiners

Ilinca, Dragos. Google. Buzz-for-Business. (2010, March). Retrieved March 19, 2010, from http://www.scribd.com/doc/27816588/Google-Buzz-for-Business

JDBC. JDBC Overview. Retrieved on 5, December 2010. Retrieved December 5, 2010. Retrieved from the Java Sun website: http://java.sun.com/products/jdbc/overview.html

JSON. Introducing JSON. Retrieved December 5, 2010. Retrieved from JSON website: http://www.json.org/

McCandless, Michael at al. Lucene in Action, 2nd ed. Manning Publications, 2010

Sanchez, D. et al. Text Knowledge Mining: An Alternative to Text Data Mining. IEEE International Conference on Data Mining Workshops (2008, December 30).

Terracotta (2010). Quartz Overview. Retrieved December 4, 2010. Retrieved from Quartz Scheduler website: http://www.quartz-scheduler.org/

White, Tom (2003). Scheduling Reoccurring Tasks in Java Applications. Retrieved December 4, 2010. Retrieved from the IBM website: http://www.ibm.com/developerworks/java/library/j-schedule.html

XPath Introduction. Retrieved December 5, 2010. Retrieved from the W3C website: http://www.w3schools.com/xpath/xpath_intro.asp

Appendices

Appendix A. Description of CDROM Contents

The accompanying CD ROM includes the following items:

- A soft copy of this report in PDF format
  /Report/Distributed Social Networking – Report.pdf

- Five projects that constitute the Distributed Social Networking project
  (Please refer to Chapter 7 for information about deployment of projects)

    - DSN-Scheduler Project
      /Project/DSN-Scheduler

        - DSN-Scheduler Database Schema
          /Project/DSN-Scheduler/DBSchema.sql

    - DSN-DataMining project
      /Project/DSN-DataMining

    - DSN-UI project
      /Project/DSN-UI

    - DSN-Commons project
      /Project/DSN-Commons

    - OneStop Web Application
      /Project/OneStop

        - OneStop Database Schema
          /Project/OneStop/DBSchema.sql