A University-Oriented Web 2.0 Services Portal[1]

Jia Zhang, Karthik Akula, Momtazul Karim, Raghu Kumar Reddy Ariga

Department of Computer Science, Northern Illinois University, USA

jiazhang@cs.niu.edu

Abstract

This paper reports several key challenges and solutions when we apply Web 2.0 mashup technology to build a university-oriented services portal. A two-layer mashup service model is proposed as the underlying basis to support multiple granularities of services mashup. We explore a caching technique to facilitate personalizable services requests. We also report our study of mashing up Facebook as a social relationship data source.

Keywords: Web 2.0, mashup portal, Facebook, social network

1. Introduction

For a college student, there is a relatively stable set of Internet-based services he/she accesses on the daily basis. For example, a student accesses his/her email box constantly for course-related information that may come from both instructors and group project teammates. Meanwhile, the student goes to every registered course Web site (maybe via some course management system (CMS) such as Blackboard system [1]) to view course news, lecture notes, newly published homework and projects, and frequently asked questions and answers. Furthermore, the student often accesses university Web site to view school news. Moreover, the student may also check some other websites (e.g., weather forecast sites, news sites, and social networks) on the daily basis.

The motivation of our project is that: the current business model has issues with supporting the above business logic. On each day, students have to manually launch every one of these websites individually. There lacks a centralized portal where students can seamlessly access various information sources at one place.

This project aims to build a centralized services portal oriented to university students. Instead of building a fixed set of services for all students, our ultimate goal is to construct a platform to allow students to easily personalize and build interested services from various information and services resources, internal and external of universities, by themselves. Since

[1] This paper is a significant extension to our conference version: J. Zhang, M. Karim, K. Akula, and R.K.R. Ariga, "Design and Development of A University-Oriented Personalizable Web 2.0 Mashup Portal", Proceedings of IEEE International Conference on Web Services (ICWS 2008), Beijing, China, September 23-26, 2008, pp. 417-424.

providing personalizability and flexibility is our key intension, we decided to explore Web services and Web 2.0 mashup technologies to build the portal.

A Web service is a programmable software module that is equipped with standard interface descriptions and can be universally accessed through standard network communication protocols [2]. The Web services technology provides a comprehensive set of standards, including languages, protocols, and frameworks, to allow software applications to be published as machine discoverable and understandable Web services on the Internet and allow existing Web services to be easily incorporated and integrated, that is, be mashed up [2] to build new business services.

Many existing Web applications have become Web services compatible, which means that they provide Web services interfaces in addition to their normal Web interfaces. For example, popular weather report websites (e.g., Google) usually provide Web services interfaces. Therefore, users may write code to access the applications through their Web services interfaces without being forced to navigate through Web pages. Moreover, users may create new services by building workflows among multiple Web services. For example, HousingMaps.com mashes up two Web services: craigslist and Google Maps [3].

As the first step, we conducted a series of surveys at Northern Illinois University (NIU) to identify an initial set of Internet services that NIU students tend to exploit on a daily basis; then we explored the plausibility of using the aforementioned technologies to construct the services portal: iNIU. In this paper, we report our strategies and solutions to address four key technical challenges that we encountered in the process of design and development of iNIU.

The first issue is how to use mashup technology to allow students to utilize existing similar services to obtain a more comprehensive service. Multiple service providers may offer the same functionality with different emphases. Thus, a service receiver may have to integrate their data to obtain a comprehensive view. Let us take weather report Web services as an example. Several popular Web services exist, such as Google weather report and Global weather report. They both provide satisfactory weather information. However, there are some differences between them: Yahoo weather provides weather image while Global weather provides pressure information. To obtain comprehensive weather data, a student may have to visit both service sites and manually integrate the information. Our services portal explores to analyze the differences between multiple services and allow students to seamlessly integrate services (i.e., a higher layer of services mashup) and personalize their preferences over different services. We propose a two-layer mashup service model to facilitate different granularities of services integration and mashup.

The second issue is how to build an integrated social network for students. Multiple social networks exist nowadays represented by Facebook [4] and OpenSocial [5]. These social networks provide similar functionalities to allow people to make friends and communicate with each other over the Internet. However, social relationships contained in different social networks cannot be shared with each other. Our surveys found that students typically simultaneously join multiple social networks. As a result, students have to sign in multiple social networks to interact with different social groups. In this project, we explore to retrieve social relationships from existing social networks (i.e., Facebook).

The third issue is how to enhance performance of mashed-up services. Our surveys reveal that many students access the same services every day, for example, they check local weather forecast. Since these students come from the same location, it is much more efficient that the services portal caches the weather data to serve students with the same requests. Services portal may refresh the local cache periodically. Meanwhile, because students may establish different preferences over different services, whether caching finalized services request results or individual services data needs to be decided.

Fourth, one unique feature of the Web services paradigm is its remote hosting. Instead of being deployed and installed on a local environment, a Web service is hosted by its service provider. Therefore, Due to unpredictable network conditions, it is possible that a reliable Web service may become unavailable at some time points. Therefore, backup services that may become substitutes should be taken into consideration in our services portal. In addition, we allow students to prioritize candidate services based on their own preferences.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we present our designed surveys and obtained results. In Section 4, we present our layered mashup service model. In Section 5, we present our services-oriented caching mechanism. In Section 6, we discuss our Facebook connection service. In Section 7, we present the overall architecture of iNIU. In Section 8, we make conclusions.

2. Related work

Mashup has been considered as an important technique for *ad hoc* Web services integration. For example, HousingMaps.com [3] is an early practice of mashing up two Web services: craigslist and Google Maps. As another example, Programmableweb.com [6] provides a mashup registry for over 2000 mashup services.

Meanwhile, a number of mashup development tools have been produced, such as Google Mashup Editor [7], IBM QEDWiki [8], Microsoft PopFly [9], and Intel Mash Maker [10]. These tools provide visual editing platforms for users to integrate data elements from various resources and create new mashup services. In contrast with these tools that provide generic mashup editing platforms, our services portal intends to provide a university students-oriented mashup and services provisioning environment, with an underlying mashup service model to guide service integration and mashup. In addition, we focus on enabling a high level of services mashup that integrates multiple services instead of merely mixing them. Moreover, we focus on portal caching for enhancing services provisioning performance. Furthermore, we explore how to use existing social networks as information resources.

Social networks aim to provide a platform and environment for people to facilitate communication and share of information with friends, family members and colleagues. The underlying framework is a social network that reflects a digital mapping of people's real-world social connections. Example social networks are Facebook [4] and OpenSocial [5]. We chose to explore how to utilize Facebook as a social relationship source since it is the most popular social network platform in US academic world.

myNIU [11] was launched in 2008 to provide a personalizable portal serving NIU faculties and students. However, it provides a fixed set of services based on information sources managed by PeopleSoft software. External services cannot be integrated into the portal. In contrast to myNIU, iNIU intends to allow students to mashup internal and external services. At the near future, we also seek to utilize myNIU as one service base

We reported the overall architecture and implemented services of iNIU in our paper participated in the 2008 global student contest on Services Computing [12]. In contrast to our previous work introduces the system functions, this paper focuses on our detailed design and solutions to four major technical challenges we encountered in the process of building iNIU.

3. Survey and results analysis

We designed and conducted a series of surveys, which main purpose is to identify students' commonly used online services to form an initial candidate list for iNIU. The surveys were conducted at two classes in the Department of Computer Science at NIU in Spring 2008. We chose one class at the undergraduate level and the other one at the graduate level, so that we could obtain students' interests in a broader range. 32 students participated in the survey.

The surveys contain a set of multiple-choice questions. Table 1 summarizes the survey questions, observations, and conclusions. Note that one row in Table 1 might represent a combination of several related survey questions.

Question 1 intends to identify the most commonly used online services by students. The top five services are: email, Instant Messenger (IM), weather, social networks, and news.

Question 2 intends to investigate how students would like to access course information. Our results show three channels: centralized portals (e.g., some course management systems (CMS)

Table 1. Survey results.

No	Question	Observations	Conclusions
1	commonly used online services	top 5: email, IM, weather, social networks, news	Students use some online services on the daily basis.
2	ways to access course information	centralized portal, course sites, instructor emails	Students prefer a centralized portal to access all course information.
3	centralized student portal	all want it	Students prefer a centralized portal to access common services.
4	centralized portal examples	igoogle, myNiu, Blackboard, yodlee.com	Need for a student-central portal.
5	features of a centralized portal	single sign-on, personalize services	Two features are mostly expected.
6	ways to access online news	centralized news portal, individual websites, combination of both	A personalizable centralized news portal is preferred.
7	news sources	cnn, google, yahoo, msnbc, northern star	Popular news sources.
8	priority of news	news topic, news source, news search	Top news personalization options.
9	account in social networks	most have accounts	Students use social networks often.
10	preferred social networks	Facebook, orkut, myspace	Need to incorporate different social networks. iNIU can start with Facebook.
11	Facebook usages	send messages, find old friends, chat with friends, make new friends	Communication with friends is one more important reason.
12	features used in Facebook	news feed, groups, status, chat features	Features to be considered in iNIU.

such as Blackboard), individual course websites, and instructor emails. Although a CMS provides a single sign-on site for students to access multiple course information, its limitation is its inflexibility since instructors have to use the CMS to manage all course information. Moreover, each CMS instance has its time frame. When a class is over, its CMS instance is not accessible any longer. Thus, many instructors still prefer to manage their own course websites. Email is another way for instructors to send messages to students. However, it lacks the ability to store communication history in a systematic manner. Therefore, we seek a light weight approach to allow instructors to use a centralized portal to distribute course information.

Question 3 intends to find out whether students would be interested in a centralized portal oriented to students. The answer is quite positive.

Question 4 intends to survey whether there exist some available centralized services portals. Students list iGoogle as a centralized news portal, myNIU as a centralized NIU portal, Blackboard as a centralized course portal, and yodlee.com as an online banking service center. As shown by the results from Question 1, however, students' commonly used services include both university and external services. None of these existing portals allows students to personalize and customize their various types of services across university boundaries.

Question 5 intends to explore what students consider the most important features of a centralized portal. Results show that two features are mostly expected: single sign-on and personalization ability. Students wish to use a single set of access code to navigate through different services. Meanwhile, students wish to be able to personalize the portal.

Question 6 intends to check how students typically view everyday news. We found that they either use some centralized news portal (e.g., google), or check individual websites, or use a combination of both. We conclude that a personalizable centralized news portal is preferred.

Question 7 intends to identify some popular news sources. The top five sources are: Google.com, Yahoo.com, MSNBC.com, and Northern Star (NIU news).

Question 8 intends to find out how students would like to sort news data. The results show that students first prefer to sorting news by topics, then to sorting news by corresponding news sources. They also would like to see an ability that allows them to perform some news search.

Question 9-12 intend to find out how students use social networks. The results show that students use social networks often, mainly for the purpose of sending messages, finding old friends, chatting with friends, and making new friends. Their mostly used social networks are: Facebook, orkut, and myspace. Especially for undergraduate students, Facebook is their favorite social network. This phenomenon is easy to be understood, as Facebook has been the most popular social network in US academia for years. The students also identify their mostly used Facebook features: news feed, groups, status, and chat features.

Our survey results provide guidance for us to design the functionalities of iNIU. As the first step, we intend to construct a prototype platform comprising news services, weather services, social networks, and several students-oriented services. More importantly, we explore to build an underlying model for the portal.

4. Mashup service model

4.1. Layered mashup service model

Based on our survey results, we identified two layers for mashup services, as shown in Figure 1. The lower level of mashup services refers to an intermingle level providing a centralized place to mix services coming from different service sources, without changing individual services. The higher level of mashup services refer to an integrated level representing new services, which may take portions from different services and incorporate them. As shown in Figure 1, the original services might not be able to be identified from a new integrated service.

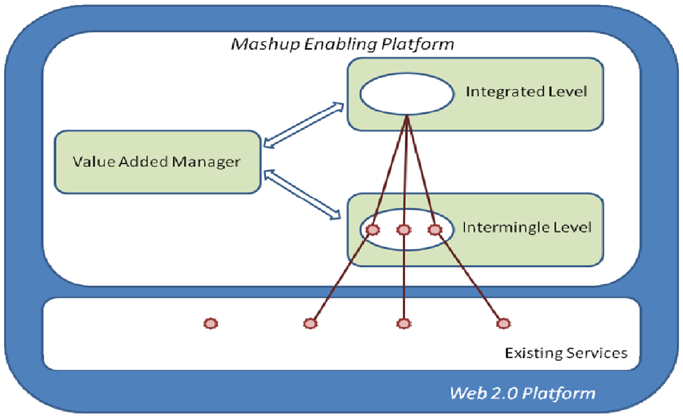
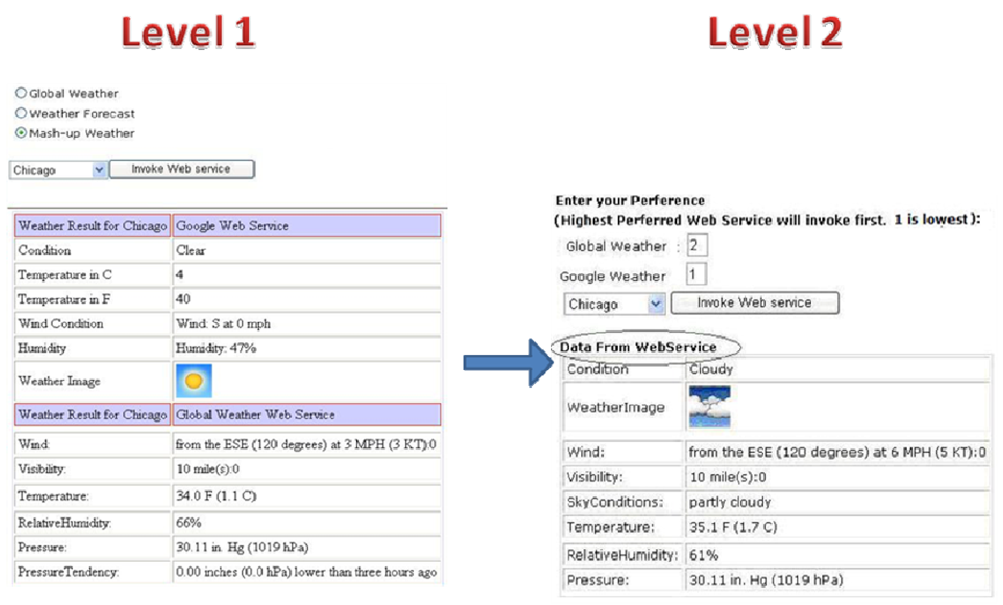


Figure 1. Layered mashup service model.

As an example, Figure 2(a) shows a level-one mashup weather service that displays the weather information for Chicago at some time point from two weather sources: Google weather service and Global weather service. As shown in Figure 2(a), different services may provide values of slightly different attributes. For example, Google weather service provides links for people to download weather images, while Global weather service does not. Taking the attribute "wind" as another example, Global weather service provides more precise wind direction than Google service provides.

A level-two mashup service may be more desirable sometimes, when the results from various service sources are integrated to provide a more comprehensive result. Figure 2(b) shows the result of an integrated level mashup service, as a counterpart of that shown in Figure 2(b). Weather data from the two sources are integrated to generate one single set of information. No duplicated attributes are shown. For example, in Figure 2(a), both Google weather service and Global weather service provide data for attribute humidity. In Figure 2(b), only one humidity item is shown from Global weather service, based on the corresponding user's preference (the user ranks Global weather service with a higher preference).



Weather Result for Chicago	Google Web Service
Condition	Clear
Temperature in C	4
Temperature in F	40
Wind Condition	Wind: S at 0 mph
Humidity	Humidity: 47%
Weather Image	
Weather Result for Chicago	Global Weather Web Service
Wind:	from the ESE (120 degrees) at 3 MPH (3 KT):0
Visibility:	10 mile(s):0
Temperature:	34.0 F (1.1 C)
RelativeHumidity:	66%
Pressure:	30.11 in. Hg (1019 hPa)
PressureTendency:	0.00 inches (0.0 hPa) lower than three hours ago

Enter your Perference
(Highest Perferred Web Service will invoke first. 1 is lowest):

Global Weather :	2
Google Weather	1

Data From WebService

Condition	Cloudy
WeatherImage	
Wind:	from the ESE (120 degrees) at 6 MPH (5 KT):0
Visibility:	10 mile(s):0
SkyConditions:	partly cloudy
Temperature:	35.1 F (1.7 C)
RelativeHumidity:	61%
Pressure:	30.11 in. Hg (1019 hPa)

(a) (b)

Figure 2. (a) Level-one weather mashup service; (b) Level-two weather mashup service.

The aggregated attribute set can be predefined by users. For a specific attribute, if multiple service sources provide different values, then user-specified preferences is taken into account to help the selection process.

4.2. Problem modeling

Using the two-layer mashup service model, a weather service may allow a user to personalize his/her preferences. A user may choose to configure to retrieve from a single Web service source or request aggregated information from multiple Web service sources. In this section, we focus on discussing how to build a level-two mashup service.

We formalize the problem as follows. Given a set of n services that provide similar functions,

$$W = \{w_1, w_2, \ldots w_n\}$$

One of its subset is defined as below.

$$W_s \subseteq W,\ W_s = \{w_{s,1}, w_{s,2}, \ldots w_{s,m}\}, m \le n$$

$$\forall w_{s,i}, 1 \le i \le m, \exists w_j \in W, 1 \le j \le n, \Rightarrow w_{s,i} = w_j$$

The subset represents an ordered set, given a predefined function *r*.

$$r(w_{s,a}) > r(w_{s,b}),\ 1 \le a < b \le m$$

The ordered set represents users' preferences ranked over the selected subset of services. The algorithm of finding an available Web service (Algorithm 1) can be illustrated in the following pseudo code:

```
$findWS - A\lg 1()$
  $loop\ (w_{s,1} - w_{s,m})$
    $if\ (w_{s,i}\ available)$
      $return\ w_{s,i};$
  $return\ fail;$
```

Looping through the ordered Web services list, the first available Web service is returned. If no Web service is available at all, a failure message is returned instead. In short, Algorithm 1 allows us to retrieve the value of an available Web service according to a user's predefined preferences.

Given a set of predefined attributes that can be applied to the set of Web services, each service is partitioned into a set of proprietary values. Another set T is used to represent whether a service provides information for each attribute. "1" means a yes and "0" means a no.

$$A = \{a_1, a_2, ...a_k\}$$
$$w \mid A = \{a_{1,w}, a_{2,w}, ...a_{k,w}\}$$
$$T = \{t_1, t_2, ...t_k\}, t_l = 1, if\ a_{l,w} \neq null$$

Algorithm 1 can be refined into Algorithm 2 as follows to show how to generate a level-2 mashup service result.

```
$findWS - A\lg 2()$
  $loop(w_{s,1} - w_{s,m})$
    $if(w_{s,i}\ available)$
      $loop\ (loop\ a_1 - a_k)$
        $if\ (!a_{j,w_{s,i}})\ findWS(a_j \mid T);$
  $return\ fail;$
```

After finding an available service according to a user's preferences, each attribute is examined. If no value is obtained, other services in the preference list that provides values for the attribute are checked sequentially, until a value is obtained. Figure 2(b) shows the result of running Algorithm 2 for the weather mashup service.

5. Services-oriented caching

5.1 Caching framework

Caching is a well-known technique to improve performance and enable faster services. Originally, a cache refers to a smaller and faster memory used to store frequently accessed data. Numerous caching strategies have been proposed at various granularities, either hardware or software level. However, services-oriented caching support has not caught sufficient attentions. With the advance of the Web 2.0 technology, the urgency regarding services-oriented caching has been greatly increased. For example, our survey reveals that NIU students typically check only one weather report site on the daily basis, even if they are aware that different sites may provide more precise data in different aspects (e.g., Google weather service provides weather image links, while Global weather service does not). Using Web 2.0, iNIU allows the students to freely mash up multiple weather report services, as shown in Figure 2(b). Assuming that each student configures three such sites to obtain a more comprehensive view, NIU students will generate extra 50,000 (2*25000 students) service calls every day for this service alone.

Recall that the main goal of iNIU is to support around 25,000 NIU students who share similar contexts. Take the iNIU weather service as an example. It is reasonable to expect that by average, every NIU student may check at least once the weather status of DeKalb, at which NIU locates. If iNIU caches local copies, we may expect a significantly less hits to the corresponding weather report service sites. Therefore, in this project we established a services-oriented caching model at iNIU to ensure higher performance of services, while studying issues related to services-oriented caching.

Figure 3 shows the overall architecture of our services caching framework. It contains eleven major components: Interface, Client Interpreter, Cache Controller, Cache Finder, Gateway, Cache Database, Cache Query Manager, Cache Updater, Result Composer, XML Parser, and Gateway.

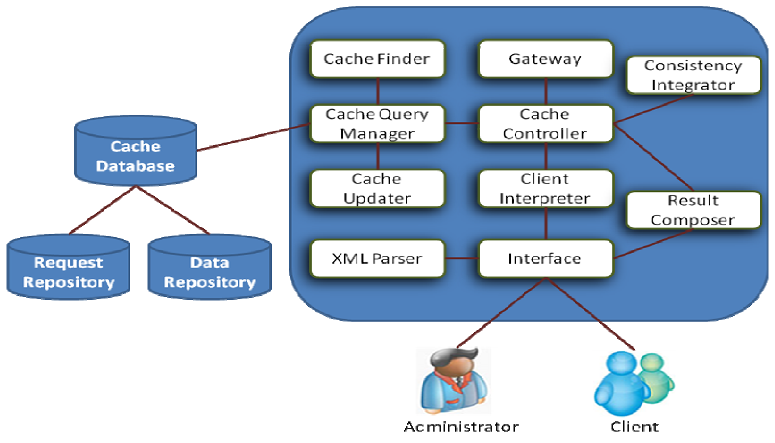


Figure 3. Caching framework at service proxy.

Cache Controller is the headquarter of the caching framework that coordinates the operations of other components. Gateway handles the communication between our caching framework and external services registries. Interface provides connection points with iNIU users, either administrators or normal users. Client Interpreter is directly connected to Interface. It interprets client requests and sends them to Cache Controller for processing. Client Interpreter is equipped with a Hash Calculator, which examines a client request and identifies comprising data (e.g., actual request (HTTP request), requested server, time, and date) and calculates a hash value. Cache Query Manager provides an interface for Cache Database and handles all operations on it (i.e., read, write, update) as instructed by Cache Controller. It hides the complexity of Cache Database. It also coordinates with Cache Finder and Cache Updater.

Cache Database is a local repository that stores temporary data. To further enhance performance, we divided it into two parts: a data repository and a request repository. The former stores actual cached services data; the latter stores conducted queries. Cache Finder reaches a hash value from Client Interpreter and searches it from Cache Database entries. If Cache Finder cannot find an item in the local cache, it forwards the request to Cache Controller that will search from the external network (i.e., Internet) through Gateway. If data are found from external services, Cache Controller stores the requests, results, and connections into Cache Database through Cache Query Manager. Then Cache Controller sends the results back to users through Client Interpreter and Result Composer. Recall that Client Interpreter separates client requests into different requests, each producing corresponding search results. Result Composer is in charge of aggregating search results coming back from different requests into one results and sending back to Interface.

Consistency Integrator is in charge of detecting and deciding whether some cached data can be used or should be considered out of date. In our design, Consistency Integrator adopts an active strategy by actively checking cached data. Cache Updater updates cache database through Cache Query Manager.

5.2 Caching management

Figure 4 shows the core objects and relationships considered in our services-oriented caching management. Our purpose is to enhance caching performance by facilitating user query in cached data. As shown in Figure 4, we cache not only services data but also queries. All cached services data have their corresponding queries stored in Request Repository. If a piece of services data is considered as out of date, its associated query will be removed as well (e.g., marked as dirty). When Client Interpreter identifies a new query, it initiates Cache Finder to search Request Repository through Cache Query Manager. As shown in Figure 4, we adopt the three core data types defined in UDDI specifications (business, service, and service type) to further enhance caching performance. Duration time (when cached data are considered stale) is associated with service type and can be configured and re-configured.

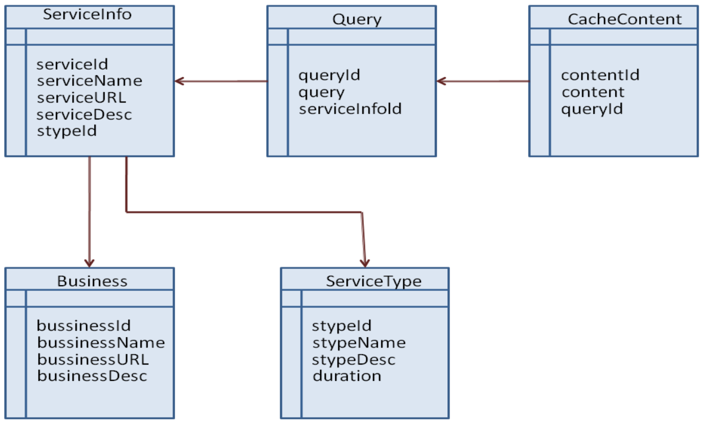
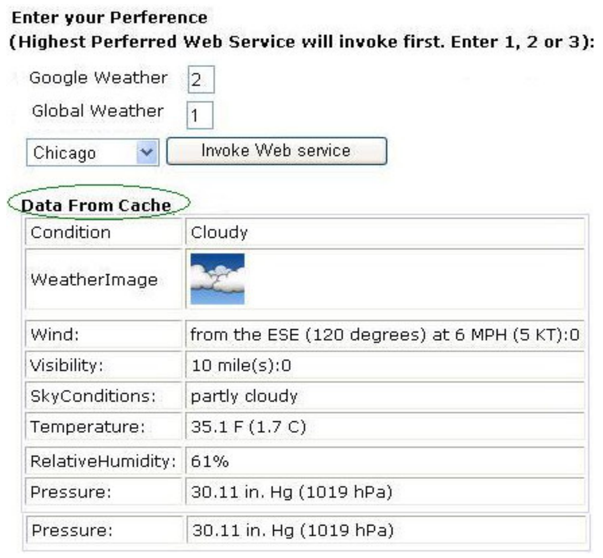


Figure 4. Caching management.

The following segment of pseudo code finds an available Web service either from the original service sources or from the cache.

```
findWS()
  if (cache & valid) return cache;
  loop ($w_{s,1} - w_{s,m}$)
    if ($w_{s,i}$ available)
      cache $w_{s,i}$; return $w_{s,i}$;
  return fail;
```

Since we allow students to configure their preferences for individual services in a mashup service, we cache query results of individual queries instead of mashed up services. Using the weather mashup service again as an example, we cache weather reports from global weather service and Google weather service as two separate items in the Data Repository shown in Figure 4. Consider another student requests a weather service before cached data become stale. Assume that his/her preferences over the two external weather services may be different (favoring Google weather service more as shown in Figure 5 compared to favoring global weather service more as shown in Figure 2(b)). Our caching service retrieves the cached weather information from the two services and dynamically mashes them up to serve students. Figure 5 illustrates that the data come from the cache repository.


Enter your Perference
(Highest Perferred Web Service will invoke first. Enter 1, 2 or 3):

Google Weather	2
Global Weather	1

Data From Cache

Condition	Cloudy
WeatherImage	
Wind:	from the ESE (120 degrees) at 6 MPH (5 KT):0
Visibility:	10 mile(s):0
SkyConditions:	partly cloudy
Temperature:	35.1 F (1.7 C)
RelativeHumidity:	61%
Pressure:	30.11 in. Hg (1019 hPa)
Pressure:	30.11 in. Hg (1019 hPa)

Figure 5. Mashed up service from cache.

6. Social network mashup

In this section, we discuss how iNIU mashes up social networking services. Social networks have been widely used by university students to conduct social activities, internal and external of university boundaries. To date, many social networking websites exist, each exhibiting some specific features. In this project, we studied the top 11 social networking websites, based on the user count on September 12, 2007.

6.1 Social network comparison

The term of social network was coined by J.A. Barnes in 1954, to represent an association of people drawn together by family, work, or hobby [13]. While the size of a social network was defined as a group of about 100 to 150 people in 1950s, the Internet enables online social networks comprising millions of members. In recent years, a number of social networking websites have been developed to provide a virtual community for people with common interests to "hang out" together. Members create their own online profiles with biographical data, pictures, likes, dislikes and any other information. They communicate with each other by voice, chat, instant message, videoconference, and blogs. Such websites typically provide a set of services to support member communication and social networking.

We have conducted some preliminary research work on comparing the features of existing social networking websites. Our experiment was designed as follows. In Spring 2008, a class of student volunteers at NIU studied the most popular 11 social networking websites in a project of forming study groups using the social networks. Our studied websites include Facebook, MySpace, flickr, classmates, and so on.

A set of topics are provided for the students to select from. If a student is interested in a topic (e.g., Web 2.0), she is asked to find members who are willing to work with her on the same project. The students were asked to follow the following stepwise procedure. (1) Search for the community group (e.g., Web 2.0) on the social networking website; (2) If the community exists, then join; (3) If not, create a community; (4) Invite classmates to join the community; (5) Locate classmates who are ready to work on the project; (6) Find how to share knowledge among members in the community; and (7) Find how to communicate between members in the community in real time.

Using the same scenario, we asked the students to exercise on each of the 11 websites to form the team. Each student was asked to compare and identify features of each website, including rating, uniqueness, negative sides, and overall ranking, and so on. Our initial results are summarized in Table I and II.

Table I. Generic comparison of 11 social networking websites

URL	User Count (Mill)	Popular rank	Membership	Code open	Storage
bebo.com	34	3	Free	No	unlimited
broadcaster.com	26	-	Free	No	Unlimited
classmates.com	40	8	Free/Gold	No	Limited
cyworld.com	21	-	Free	No	Unlimited
facebook.com	48	2	Free	Yes	Unlimited
flickr.com	4	-	Free/Premium	No	100MB
friendsreunited.com	19	-	Free	No	100MB
friendster.com	50	13	Free	No	Unlimited
myspace.com	206	1	Free	No	Unlimited
orkut.com	68	14	Free	No	1GB
spaces.live.com	120	15	Free	No	1GB

Table I shows that these websites all provide free memberships with significant storage spaces. They are all popular and have a large user base. The websites typically do not provide open source except Facebook.

Table II. Usability comparison of 11 social networking websites

URL	Create Profile	Check other profiles	Create comm.	Commu-nication	Uniqueness	Negatives	Comments
bebo.com	Any email id	yes	Easy	Text messages, blogs	Have mail, music, video, authors	No groups	Good interface for profiles
broadcaster.com	Any email id	yes	Moderate	Webcam, text, audio, video	Webcam live chat, Video, Liveweb cam, music. movies	Groups restricted	Good for online webchat with text audio video
classmates.com	Any email id	Yes	Moderate	Text messages	Lot of database available	No create generic groups	Good for finding old friends
cyworld.com	Any email id	Yes	Moderate	Text, audio, video messages	Cool colors and attractive design with profile widgets	New in US	Good profile design, easy communication with members
facebook.com	Any email id	yes	Easy	Text, audio, video, mobile	Most popular among students	Personal data is open to app. Developers, privacy concern	Good interface and lots of applications
flickr.com	Yahoo id	no	No	Cannot communicate	Album view is good	No community formation, no communication	Only for photos, nothing else
friendsreunited.com	Any email id	yes	Yes	Text	Dating, jobs, forefathers, connections with friends	Poor design, few applications	Bad designing, hard to communicate
friendster.com	Any email id	yes	Yes	Text messages	Video, blog, reviews, photos, bulletinboard	Video are directed to other websites	Lots of advertisements
myspace.com	Any email id	Yes	Easy	Text, photos, audio, video	All ages use this	Lots of advertisements	Almost every feature needed
orkut.com	Google id	Yes Limited	Easy	Text	Forums in communities good features	Not popular in USA, no applications	Simple design and linked with youtube.com
spaces.live.com	Hotmail or Live id	No	Hard	Through text messages	Good look	New, not popular	Offers skydrive to store 1GB data

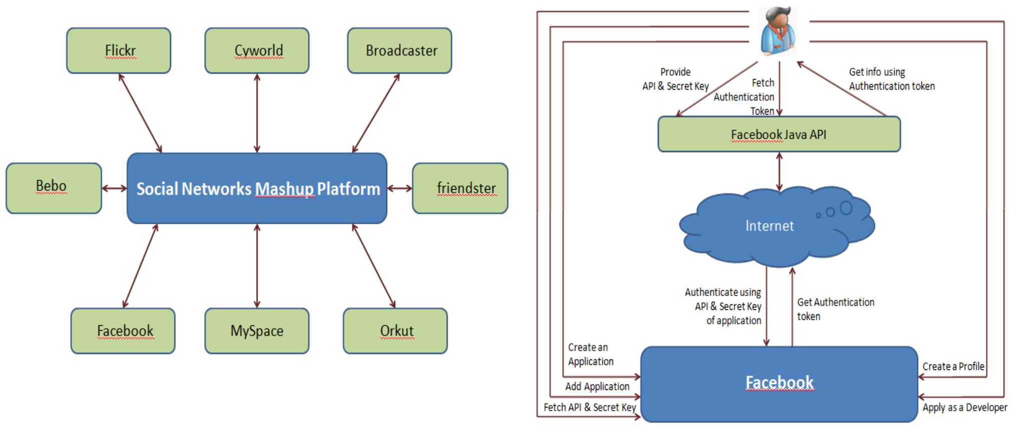
Table II reveals several facts. All these websites allow users to create their personal profiles. Most sites allow users view their friends' profiles marked as public. Their abilities to create communities vary; and their provided channels for communication vary as well. Each site shows some unique features, such as whether they can handle emails, video, images, and so on. For every website, our students also identified some shortcomings that they do not like. For example, MySpace has too many advertisements that may distract users. Finally, we summarized the students' overall comments for each website.

In summary, as shown in Table I and II, all social networking websites provide similar functionalities to allow people to make friends and communicate with each other over the Internet. Every website provides some unique and interesting features. However, each one shows some limitations, from user perspective. That may explain why no social network website dominates. Our survey also proves that students typically join multiple social networks simultaneously. As a result, students have to sign in multiple social networks to interact with different social groups.

6.2 Challenges for social network mashup

Students create multiple accounts in various social networks that do not interoperate and interact. As a result, the social relationships contained and maintained in different social networks cannot be shared with each other. In our experiment of forming study groups, a student may have to form two simultaneous communities in multiple social networks, because some of classmates only have accounts in one social network while others only have accounts in another social network.

Therefore, in this project, we aim to explore how to adopt the Web 2.0 technology to build a social networks mashup platform. As shown in Figure 6(a), such a platform shall be able to retrieve social relationships from existing social networks (e.g., Facebook, MySpace, and Orkut) and allow users to exploit these data to create new business services (e.g., to form a study group as in our experiment).



(a) (b)

Figure 6. (a) Proposed social network mashup infrastructure; (b) Facebook invocation

Our idea is to treat all of these social networks as common data sources, so that their comprising social relationships can be extracted, integrated, and utilized in the iNIU environment. In other words, using the Web 2.0 technology, iNIU mashes up existing social networks and provide further support for social connections. Then our challenges turn into how to allow iNIU to automatically analyze and extract social relationships stored in various social networks.

Retrieving social relationships from various social networks is not a trivial task, even though a social network website provides programming API such as Facebook. Figure 6(b) shows how a user can access Facebook data via Facebook Java API. A user has to first create an account in Facebook. Then the user has to apply to Facebook as a developer. Upon acceptance, the user can create an application in the Facebook platform and apply to be listed as a plug-in. Upon approval, a unique application-specific API key and secret key will be granted by Facebook. These two keys are necessary for accessing Facebook through its Java API. In other words, a program can only access Facebook through an approved plug-in application. The user also has to add the application into the profile of the account before the API and secret keys can be used. Before a Facebook API method can be invoked, as shown in Figure 6(b), the user has to log into Facebook and maintain an active session, whose unique authentication token can be obtained and used to invoke Facebook API calls, together with the API key and secret key.

6.3 Connection to Facebook

Our first step is to study whether social relationships can be extracted out of existing social networks and construct a prototype system. As a starting point, we focus on Facebook. To date, Facebook [4] is considered the most popular online social network for the US university students. As revealed by our survey results, most NIU students have accounts in Facebook and they use Facebook very frequently, if not on the daily basis, to communicate with their friends, inside and outside of school boundary. Thus, we decided to start from Facebook and explore how to exploit the knowledge stored in Facebook to enhance social networking at iNIU. For example, we intend to allow NIU students to find other NIU students who have accounts in Facebook, by mining the relationships stored in Facebook.

Facebook provides a set of Representational State Transfer (REST)-based Application Programming Interfaces (APIs), meaning that Facebook method calls can be made over the Internet by sending HTTP GET or POST requests to the Facebook REST server. However, a significant technical challenge exists. Currently, Facebook does not provide APIs for external applications to invoke its functionalities directly. All applications can only be built on top of the Facebook platform as plug-ins, as shown in Figure 7. Facebook users must find a specific application listed in the available application list and utilize the application.

For simplicity reason and as a starting point, we intend to extract the personal profile of a Facebook user, such as the user's personal information and all friends in the context of Facebook. Our goal is to build a service with two functions. On one hand, it can automatically extract information from Facebook. On the other hand, it can allow other programs to use the information to develop new services.

For experimental purpose, we registered a developer application at Facebook named "Personalized Profile Viewer," as shown in Figure 7. Our method is a workaround to access Facebook from a remote machine, where iNIU resides. Two major components enable the connection service: a Facebook REST client and a Facebook engine. An instance of Facebook REST client consists of a set of parameters (such as api_key, session_key, and call_id) that can be used to create a session for users to access Facebook plug-in applications. Facebook engine is a Java servlet residing on iNIU server. It uses an instance of Facebook REST client java class to access profiles residing on Facebook platform with the URL as:

```
public static final String FB_SERVER = "api.facebook.com/restserver.php";
public static final String SERVER_ADDR = "http://" + FB_SERVER;
public static final String HTTPS_SERVER_ADDR = "https://" + FB_SERVER;
```

Our strategy and implementation are summarized in Figure 7. We will walk users through our algorithm and procedure.

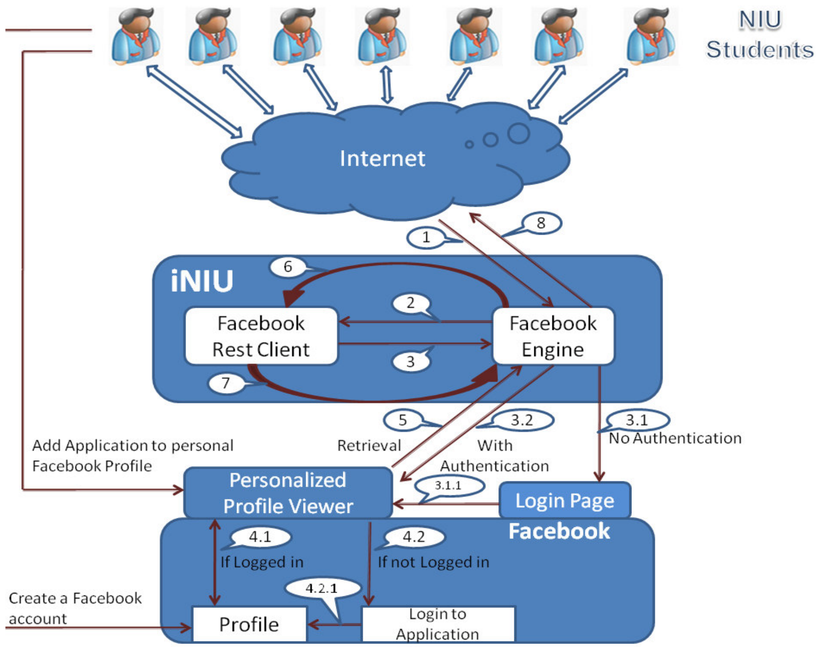


Figure 7. Communication protocol to Facebook.

In Step 1, NIU students send requests to iNIU regarding Facebook information through the Internet. Note that with this solution, NIU students have to hold a Facebook account before exploiting our Facebook connection service. Besides Facebook accounts, students may or may not add our "Personalized Profile Viewer" application to their Facebook accounts.

In Step 2, iNIU initiates two components to enable the communication to Facebook: a Facebook engine and a Facebook REST client. The former checks for authentication token by using the latter.

In Step 3, Facebook REST client checks whether there exist running sessions on Facebook for the student who submits the request. If it does not exist, in Step 3.1, Facebook engine redirects the student to Facebook login page. After logging into Facebook, the student may decide to add "Personalized Profile Viewer" application to his/her profile and move to "Personalized Profile Viewer" in Step 3.1.1. If running sessions exist, it means that the security token for the student is available. As shown in Step 3.2, Facebook engine directly directs the student to "Personalized Profile Viewer."

In Step 4, Facebook checks whether the student has logged onto their website. If already logged in, as shown in Step 4.1, our "Personalized Profile Viewer" retrieves the student's Facebook profile and social context by using Facebook APIs. Otherwise, as shown in Step 4.2, the student is first directed to the Facebook login page and then our "Personalized Profile Viewer" can retrieve the student's Facebook profile information.

In Step 5, our "Personalized Profile Viewer" sends back profile information to Facebook engine.

In Step 6, the data are sent back to Facebook Rest client to prepare return data.

In Step 7, Facebook Rest client returns prepared data to Facebook engine.

In Step 8, Facebook engine forwards the data through the Internet to the student.

Our iNIU Facebook engine is able to add social context to iNIU portal by utilizing profiles, friends, photos, and event data to an iNIU student as long as a student has a Facebook account and has logged into Facebook.

6.4 Facebook connection Web service

Our iNIU engine has successfully extracted data from Facebook. However, it has several limitations. First, since we used the Facebook Client APIs to fetch user profiles, a user has to log into Facebook through its login page before using our service in iNIU. In addition, an active session has to be maintained, so that we can obtain the authentication token (a unique alphanumerical number that uniquely represents the session for the user) to fetch the user's profile from Facebook server. Otherwise, the users will be redirected to Facebook login page. This prerequisite is undesirable, since iNIU shall keep transparent to users the Facebook-specific login process. Second, a user has to add our Facebook application ("Personalized Profile Viewer") to his/her Facebook profile ahead of time. If the answer is a no, our engine will redirect the user to the Facebook login page, so that the user can login and add our application. This action shall be automated transparent to users. Third, single sign on (SSO) [14] has been widely utilized in network computing to enable users to logon to the network and access resources from computers or domains where they have no accounts. We intend to eliminate the need for users to remember credentials for every social network. Instead, iNIU shall be able to manage automatic log in process on behalf of users.

Recall that Facebook requires authentication token from an active Facebook session. After experimenting various ways using Java APIs but still failed, we came across .NET APIs that allow to authenticate a user without a browser. The core section of code fetching an authentication token follows:

```
HttpSession session = request.getSession();
String sessionKey = (String) session.getAttribute("fbSession");
String fbtoken = request.getParameter("auth_token");

if (fbtoken != null) {
  fbRestClient = new FacebookXmlRestClient(apiKey, secretKey);
  try {
    sessionKey = fbRestClient.auth_getSession(fbtoken);
    session.setAttribute("fbSession",sessionKey);
  } catch (FacebookException e) {
    e.printStackTrace();
  }
}
```

We used HTTPWebRequest and HttpWebResponse to login virtually (similar to HttpUnit) in .Net. We first create a request to the Facebook login page using the GET request type, then we append the credentials to the corresponding fields of the login page (Textbox provided for username and password). By using the POST request, we post the values to the server. Thus, we can get an authentication token.

Further, we established a Web service for Facebook, as shown in Figure 8. The service is independent of our iNIU system and resides on another machine. It is implemented in .NET, while our iNIU system is implemented in Java. As shown in Figure 8, our Facebook Web service handles all communication activities with Facebook. It hides all the complicated Facebook invocation details and provides a Web service interface. Follows is the WSDL interface for retrieving all Facebook friend information by providing a user id and password.

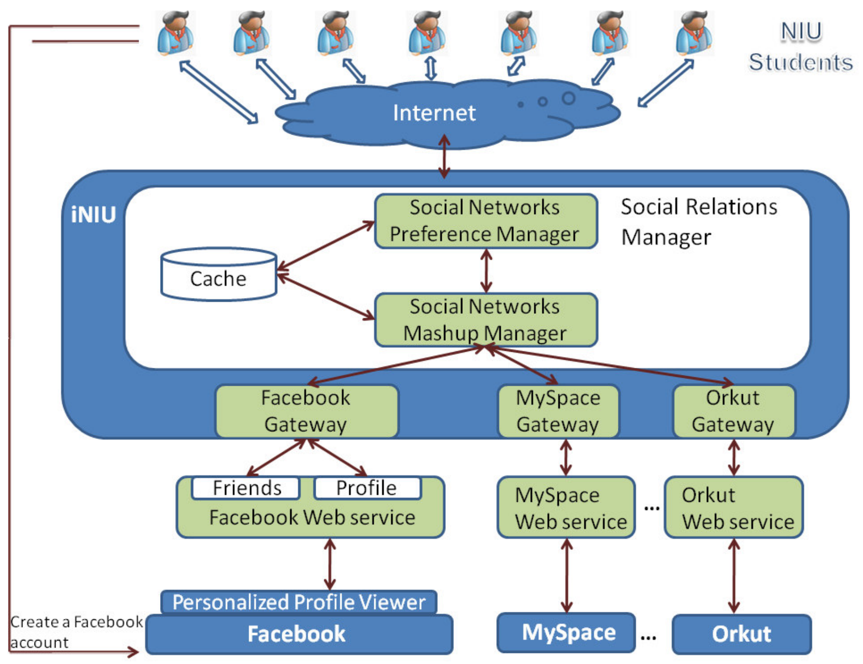


Figure 8. Facebook connection Web service.

```
<s:element name="GetFriends">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetFriendsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetFriendsResult"
type="tns:ArrayOfString"/>
    </s:sequence>
  </s:complexType>
</s:element>
…

<wsdl:operation name="GetFriends">
  <wsdl:input message="tns:GetFriendsSoapIn" />
  <wsdl:output message="tns:GetFriendsSoapOut" />
</wsdl:operation>
```

Based on the Facebook Web service constructed, inside of iNIU, we built a Facebook gateway that acts as a stub, which is a Web service client on behalf of iNIU. Similarly, iNIU builds a dedicated gateway for every social network. Such a design provides scalability if new social networks are added into the scene. As shown in Figure 8, we built a Social Relationship Manager module inside of iNIU comprising two major components: a Social Networks Mashup Manager and a Social Networks Preference Manager. The former handles two-layer social relationship mashup; the latter allows iNIU users to specify their personal preferences on utilizing social relationships from available social networks. A local cache serves to enhance service performance. The detailed techniques on layered service mashup and services-oriented caching are discussed in Sections 4 and 5.

7. System implementation

Figure 9 shows the overall architecture of the iNIU portal. The eight core components are: Web 2.0 Platform, Service Configuration Manager, Mashup Manager, Preference Manager, Cache Manager, SOAP Message Generator, SOAP Message Interpreter, and Service Invoker. As shown in Figure 9, various external Web services serve as backend data sources for iNIU: Amazon.com services, Yahoo! services, Google services, Facebook, Global weather services, and various news sources.

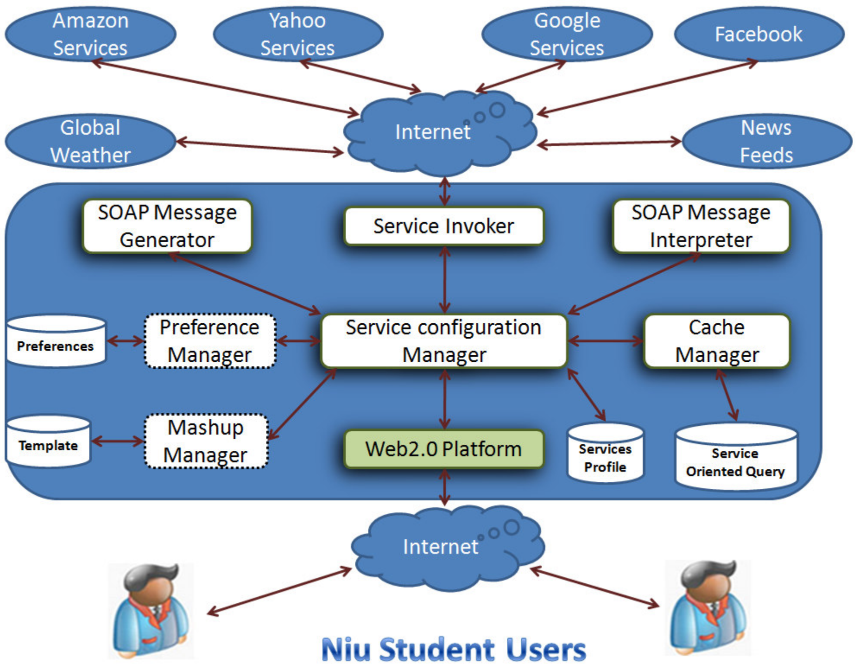


Figure 9. iNIU architecture

Web 2.0 Platform provides a user interface and mashup environment for NIU students. Service Configuration Manager coordinates all comprising components and allows students to design a personalized service portal. For example, a student may configure weather service and news service, while another one may configure weather service and social networking service. Two dotted boxes are Preference Manager and Mashup Manager. The former allows students to specify their preferences on various data sources over a specific service; the latter allows students to specify the level of mashup over a specific service. These two components combined help students to configure their personalized portal. Note that they are both generic components. For every specified service type, an instance pair will be created. For example, as shown in Figure 8, a Social Relationships Manager is a conceptual module that comprises two components: Social Networks Preference Manager and Social Networks Mashup Manager are two instances of the above two generic components.

Cache Manager provides caching service for iNIU. SOAP Message Generator creates SOAP messages. SOAP Message Interpreter analyzes incoming SOAP responses and conducts necessary parameter conversions (e.g., meter to inch). Service Invoker is in charge of communicating with external services. Four components maintain proprietary databases: Preference Manager, Mashup Manager, Service Configuration Manager, and Caching Manager.

8. Conclusions

In this paper, we reported our design and solutions to four technical challenges when we construct a university student-oriented services portal. We present a layered service model to provide mashup services at multiple granularities. We also reported our service-oriented caching mechanism, based on user preferences and service availability. Furthermore, we present our strategy of utilizing Facebook as a resource base to mashup social relationships from various social networking websites.

Our prototype of iNIU provides a proof of concept toward building a university-oriented Web 2.0 mashup services portal. We plan to continue our research in the following several directions. First, we plan to explore building an NIU-based social environment, by utilizing knowledge contained in Facebook and other social networks. Second, we plan to build a customizable news service, for students to personalize news sources as well as caching and ranking preferences. Third, we plan to publish iNIU to NIU students to perform case studies.

9. Acknowledgement

This research is supported by the Computer Science Department of Northern Illinois University.

10. References

[1] Blackboard.com, "Blackboard Learning System", Available from: http://www.blackboard.com/products/academic_suite/learning_system/index.

[2] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. Springer, 2007.

[3] HousingMaps.com, "HousingMaps.com", Available from: http://www.HousingMaps.com.

[4] Facebook, "Facebook", Available from: http://www.facebook.com/.

[5] Google, "Google Launches OpenSocial to Spread Social Applications Across the Web", 2007, Available from: http://www.google.com/intl/en/press/pressrel/opensocial.html.

[6] programmableweb.com, "programmableweb.com", Available from: http://www.programmableweb.com/.

[7] Google, "Google Mashup Editor", 2008, Available from: http://code.google.com/gme.

[8] IBM, "IBM Mashup Starter Kit", 2007, Available from: http://www.alphaworks.ibm.com/tech/ibmmsk.

[9] Microsoft, "Popfly", 2008, Available from: http://www.popfly.com/Overview.

[10] Intel, "Intel Mash Maker", 2008, Available from: http://mashmaker.intel.com/.

[11] NIU, "myMIU", 2008, Available from: http://myNIU.edu.

[12] R.K.R. Ariga, K. Akula, S.R. Gujjala, Momtazul Karim, S. Ramesh, and J. Zhang, "iNIU - A Services Portal for NIU Students", in Proceedings of *Proceedings of IEEE Congress on Services Computing (SERVICES 2008)*, Jul. 8-Jul. 11, 2008, Honolulu, Hawaii, USA, pp.

[13] J.A. Barnes, "Class and Committees in a Norwegian Island Parish", *Human Relations*, 1954, 7: pp. 39-58.

[14] I. Ristic, "Apache Security", 2005, Available.