

Design and Development of A University-Oriented Personalizable Web 2.0 Mashup Portal

Jia Zhang, Momtazul Karim, Karthik Akula, 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 preliminary practice of exploiting Facebook as a social relationship data source.

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 obvious: 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 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 connection mechanism to Facebook. In Section 7, 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 [12] that 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. 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.

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.

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) 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 construct

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

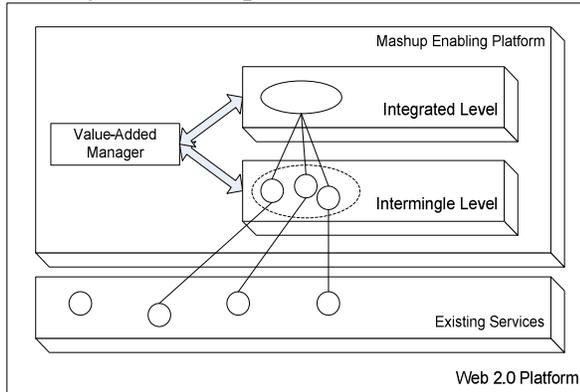


Figure 1. Layered mashup service model.

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

- Global Weather
- Weather Forecast
- Mash-up Weather

Chicago

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

Figure 2. Level-one weather mashup service.

As an example, Figure 2 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, different services may provide values of slightly different attributes. For example, Google weather service provides links for weather images to be downloaded, while Global weather service does not. Taking the attribute “wind” as another example, Global weather service provides more precise direction of wind 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 3 shows the result of an integrated level mashup service, as a counterpart of that shown in Figure 2. 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, both Google weather service and Global weather service provide data for attribute humidity. In Figure 3, 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).

Enter your Preference
Highest Preferred Web Service will invoke first. 1 is lowest):

Global Weather :
 Google Weather :
 Chicago

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)

Figure 3. 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,

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

One of its subset is defined as below.

$$W_s \subseteq W, W_s = \{w_{s,1}, w_{s,2}, \dots, w_{s,m}\}, m \leq n$$

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

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

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

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 – Alg 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.

$$A = \{a_1, a_2, \dots, a_k\}$$

$$w \mid A = \{a_{1,w}, a_{2,w}, \dots, a_{k,w}\}$$

$$T = \{t_1, t_2, \dots, t_k\}, t_l = 1, \text{ if } a_{l,w} \neq \text{null}$$

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

```

findWS – Alg 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 3 shows the result of running Algorithm 2 for the weather mashup service.

5. Service-oriented caching

5.1 Caching framework

Caching is a well-known technique to improve performance and enable faster services. Originally, a cache is a memory technique where data is stored temporarily for frequent access. Numerous caching strategies have been used at various granularities, either hardware or software level. However, service-oriented caching support has not caught sufficient attentions.

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 research we intend to establish a service-oriented caching model at iNIU to ensure higher performance of services, while studying issues related to service-oriented caching.

Figure 4 shows the overall architecture of our service 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.

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 divide it into two parts: a data repository and a request repository. The former stores actual cached services data; the latter stores conducted queries.

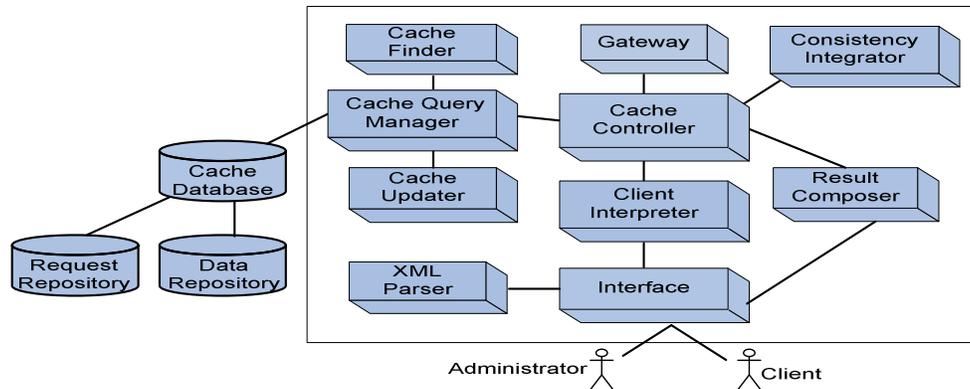


Figure 4. Caching framework at service proxy.

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

As shown in Figure 5, we cache not only services data but also queries. The purpose is to further enhance caching performance by facilitating user query in cached data. 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 5, 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.

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 (ws,1 - ws,m)
    if (ws,i available)
      cache ws,i; return ws,i;
  return fail;

```

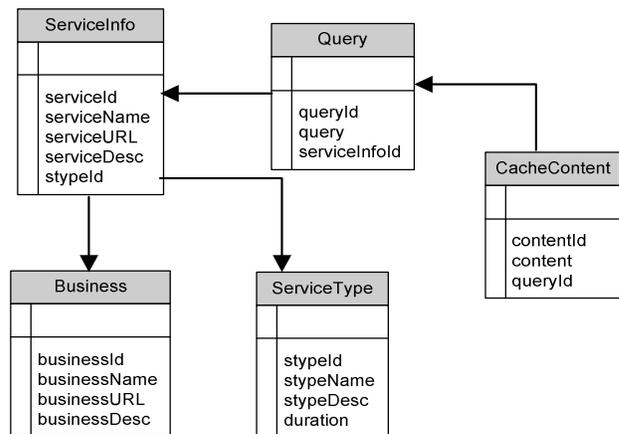


Figure 5. Caching management.

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 6 compared to favoring global weather service more as

shown in Figure 3). Our caching service retrieves the cached weather information from the two services and dynamically mashes them up to serve students. Figure 6 illustrates that the data come from the cache repository.

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

Google Weather

Global Weather

Chicago

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 6. Mashed up service from cache.

6. Connection to social networks

Facebook [4] is the most popular social network environment in the universities in the US. As shown in our survey results, many NIU students have accounts in Facebook and use Facebook frequently to communicate with friends. Recall that the major goal of iNIU is to provide a central portal that mashes up various services. Thus, we explore how to seamlessly exploit the knowledge stored in Facebook to enhance social networking at iNIU. For example, we intend to allow NIU students to find out other NIU students through Facebook relationships.

Our idea is to treat Facebook as a common data source, so that its comprising social relationships can be extracted and utilized in the iNIU environment. Facebook provides a set of APIs with a Representational State Transfer (REST)-based interface, meaning that Facebook method calls are made over the Internet by sending HTTP GET or POST requests to the Facebook REST server. The challenge is that, currently, Facebook does not provide APIs for external applications to invoke its functionalities remotely. All applications can only be built on top of the Facebook platform as plug-ins, as shown in Figure 7. As a starting point and for experimental purpose, we registered a developer application named “Personalized Profile Viewer,” as shown in Figure 7. Thus, we can test and prove our ideas through our service, without Facebook users seeing our service listed in the available application list.

Our strategy and design are illustrated 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. We use an instance of Facebook Rest client consisting 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.

We will walk users through our algorithm and procedure, as denoted in Figure 7.

In Step 1, NIU students send requests to iNIU regarding Facebook information through the Internet. Note that with our current 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 Facebook connection service 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

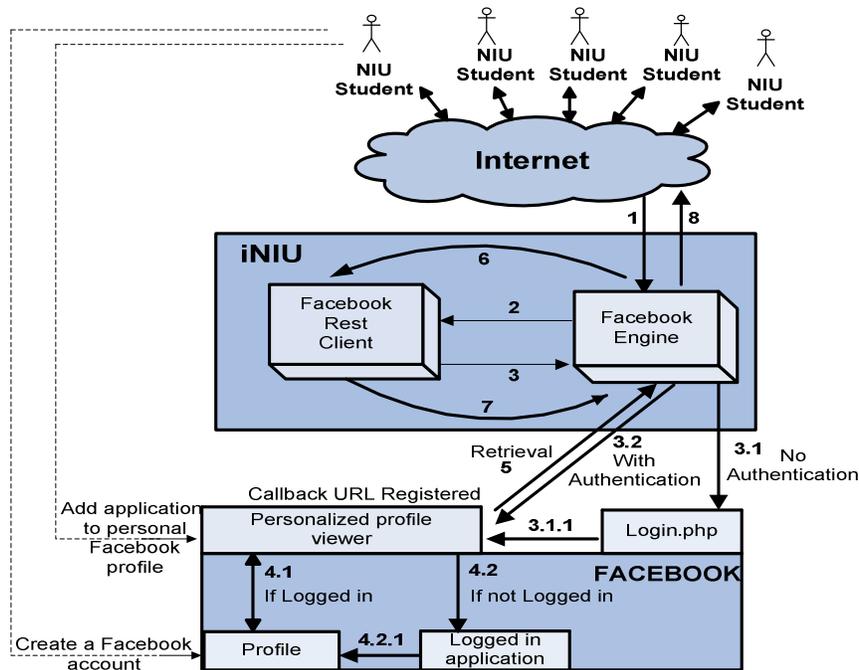


Figure 7. Communication protocol to Facebook.

student has a Facebook account and has logged into Facebook.

7. Conclusions

In this paper, we reported our design and solutions to four technical challenges. 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.

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. 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.

8. 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*. 2007: Springer.

[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/ibmmask>.
 [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.