

seMeja API Design Based on CRUD+N Concept

Marini Abu Bakar, Surya Ismail, Sufian Idris and Zarina Shukur

Faculty of Information Science and Technology, Universiti of Kebangsaan Malaysia, Selangor, Malaysia

Article history

Received: 24-10-2014

Revised: 12-03- 2015

Accepted: 27-06-2015

Corresponding Author:

Marini Abu Bakar

Universiti Kebangsaan

Malaysia, Selangor, Malaysia

Email: marini@ukm.edu.my

Abstract: seMeja is an ongoing research to develop a desktop system for university environment. To support the development of this system, an Application Programming Interface (API) is needed to provide access to online university services such as course notes, course schedules and course registration interface. This paper proposes a research to create and design the API for seMeja named seMejaAPI. The primary goal for this API is to allow programmers to quickly develop applications that can interact with university online system that support the API. The API needs to be versatile enough to encapsulate the variations of online university services and yet easy enough to be used by an application programmer. The design of the API is based on the four operations of Create, Read, Update and Delete (CRUD). In addition to these four basic concepts, the concept of 'Notify' has been added to support the registration for push-style notifications. These principles are then combined with an existing university-based ontology. This ontology defines the various objects used in a university environment. Two prototypes were then developed and tested to demonstrate an implementation of a portion of the API, along with a small working application.

Keywords: API, seMeja Desktop Environment, University Operating System, Ontology, CRUD

Introduction

The seMeja Desktop Environment is an operating system for university students running on a Linux operating system. It was designed and developed at Universiti Kebangsaan Malaysia (UKM) as a collaborative effort between several research groups. The desktop environment is meant to mimic the local Malaysian education environment.

To increase the effectiveness of the seMeja system, specialized applications can be developed to help students gain expertise and learn the required material. Application for seMeja should support some key features to make them more useful for university students. These features includes: (i) the use of student information to create an individualized, student-oriented user experience. The seMeja Desktop Environment requires a login, which means that the operating system knows the identity of the student user. This identification can be used to retrieve information on the student. When a student enters a university, the student's basic information is stored on various server-sides of the university systems. As the student progresses, more information is generated. For example, information like the student's faculty affiliation, registered courses and

grades record can all be accessed online. An effective seMeja application should take advantage of this wealth of information to create an individualized experience for the student. (ii) The innovative use of university educational resources. Modern universities provide online access to large amounts of educational resources. Materials such as course notes, lab manuals and publications are all available online. Applications for the seMeja system can take full advantage of these resources by centralizing, organising and presenting them in new and innovative ways. (iii) The use of research technologies. The various seMeja research groups have identified a few technologies that will potentially benefit university students. These technologies include natural language processing and notifications. seMeja applications should make use of these technologies.

The proposed seMeja architecture has been discussed by (Idris *et al.*, 2010). The seMeja system runs on a Linux-based netbook. The base operating system provides access to system utilities such as memory, processing and devices. The system utilities are accessed through the system libraries and applications while the user interface is managed by the GUI manager. The three tiers of the seMeja Environment make use of the system libraries and applications.

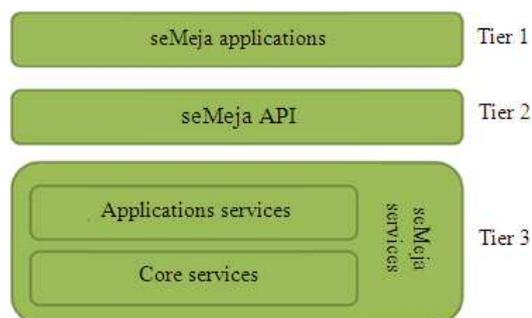


Fig. 1. Three tier architecture of the seMeja system

The three tiers are the seMeja Services layer, the seMeja API and the seMeja Applications. The seMeja system communicates with external university servers. The university servers control data related to student information and university resources.

Figure 1 presents a simplified view of the three tiers of the seMeja system. The seMeja services layer contains modules that provide useful functions for the applications. The initial design requirement divides the modules into Core Services and Application Services. The Core Services provide basic services for the desktop which are used by all applications. The Application Services provide access to domain specific services.

The seMeja API is the second tier of the seMeja Environment. Its purpose is to wrap services in a standard API that is used by developers to create applications for the seMeja Desktop Environment. The top tier of the seMeja Desktop Environment contains the seMeja applications. As mentioned in the previous section, these applications are customized for university students.

Related Works

Robillard (2009) defined an API as an interface to implemented functionality that developers can access to perform various tasks. APIs support code reuse, provide high-level abstractions that facilitate programming tasks and help unify the programming experience.

Henning (2007) proposed that APIs should be minimal. A small API, with fewer types, methods and parameters, is easier to learn, remember and use. Henning also concludes that when little is known about the context in which an API will be used, the best choice is to keep all options open and allow the API to be as widely applicable as possible. Bloch (2006) added another principle which is to not make the developer do anything the library could do. Violating this rule leads to developers having to write wrapper code that must be used every time the library is used. This type of code, known as boilerplate, is redundant and error-prone.

This section reviews existing APIs related to the seMeja API. Two existing educational systems that use APIs are discussed. In each case, the structure of the API is reviewed and the API is analysed to find key factors that make the API useful for developers.

The Sugar Toolkit

Sugar (SL, 2012) is an educational desktop environment for younger children. It was originally designed for the One Laptop per Child project. OLPC is a non-profit organization established to provide low-cost laptops to children in developing countries (OLPCF, 2012). Currently, Sugar runs on over two million laptops in 25 languages in more than 40 countries.

Sugar has an unusual user interface. The interface is designed to be comprehensible by young children, including those who are unable to read. Applications run on full screen and menus are iconic. Figure 2 shows the Sugar home view. The view changes according to the activity. The icons represent the child (the user) and his or her favourite activities. The home view changes depending on the current activity.

Developers do not develop programs using a compiled language. Instead, they develop 'Activities' using the Python programming language. Python is an interpreted language, which means by using a simple text editor, any Sugar laptop can be used to develop new Sugar 'Activities'.

Architecturally, Sugar runs on a modified GNOME Linux desktop (Fig. 3). Python is the preferred development language. Python activities access services through the Sugar toolkit. These services include a data store, clipboard capabilities and a network presence service.

Sugar is an extreme example of a customized educational Linux desktop. The seMeja Desktop Environment does not need to be as customized. However, Sugar shows the validity of the approach. Sugar runs on a Linux core and adds variety of services which can be accessed through the Sugar toolkit to create activities. This is similar to the seMeja API and applications approach used in the seMeja system.

Sugar, the API is known as the Sugar toolkit. The API has a minimal look, which reflects the ideas of simplicity and intelligibility that are part of the Sugar design philosophy. The Activity and Presence packages are good example. The Activity module provides the user interface for a Sugar activity. This allows developers to create programs with iconic menu and full screen behaviours that are essential for Sugar. The Presence package allows collaboration by providing services for native networking and information sharing.

The Moodle Activities API

Moodle is an abbreviation for Modular Object-Oriented Dynamic Learning Environment. It is an Open Source Course Management System (MC, 2012). Moodle is basically a web application. It needs to be installed on a server and it's accessed through a client browser (Fig. 4). Moodle is used in over 200 countries and has over 69,000 registered users. It is used by all sorts of organizations including primary and secondary schools, government bodies and the military.

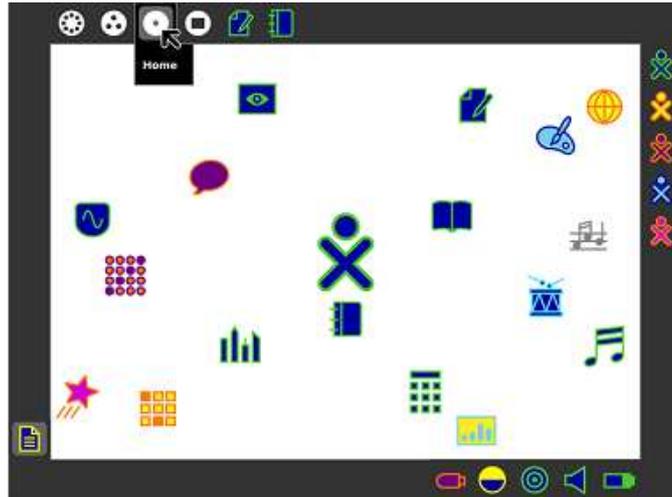


Fig. 2. The sugar home view

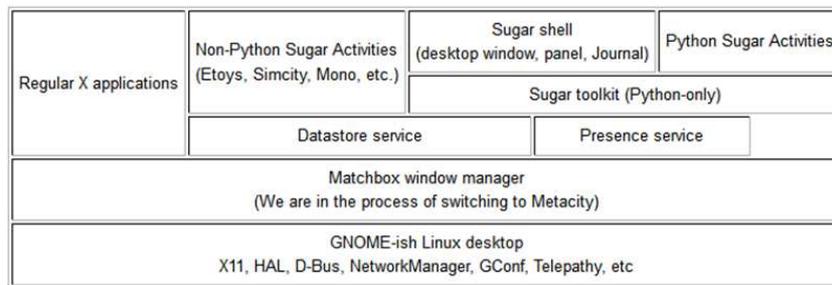


Fig. 3. Sugar architectural diagram

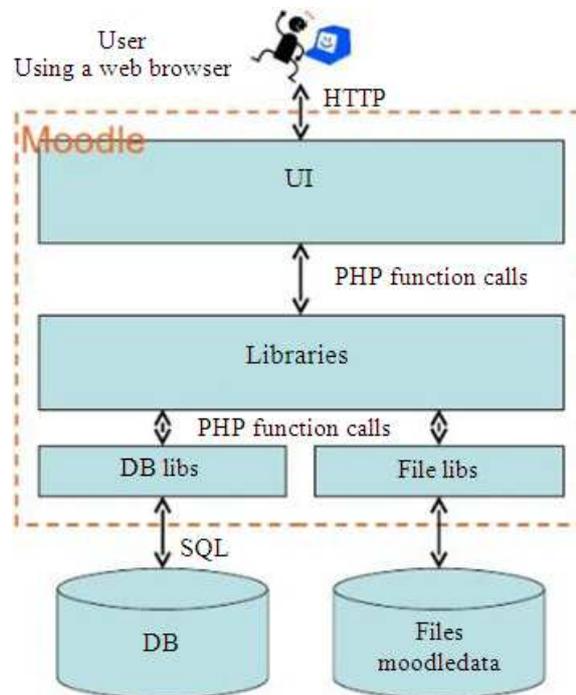


Fig. 4. Moodle as a web application

Table 1. Moodle core APIs

API	Description
Access	Determines what the current user is allowed to do.
Data manipulation	Enables read/write to databases in a consistent and safe way.
File	Controls the storage of files.
Form	Defines and handles user data via web forms.
Logging	Allows developer to add new entries to the Moodle log and define how they get displayed in reports.
Navigation	Manipulate the navigation tree to add and remove items.
Page	Used to set up the current page, add JavaScript, and configure how things will be displayed to the user.
Output	Renders the HTML for all parts of the page.
String	Get language text strings to use in the user interface, handles any language translations that might be available.
Upgrade	Controls how a module installs and upgrades itself, by keeping track of its own version.

Moodle is modular and is designed for customization. The interface is customized using themes. Themes can be applied on various levels and granularities and can be designed to suit different types of displays such as tablets and net books displays. Functionality can be customized by adding different plugins. Plugins are developed in PHP and there are hundreds already available. Plug in developers have access to basic functionality through the Core API. Table 1 shows the most frequently used Core API modules.

Moodle is different from the seMeja Desktop Environment in that it is a web-based solution, not an operating system. However, the Moodle Activity API bears many similarities to the seMejaAPI. The Activity API abstracts the functionality of class-based activities and makes them easily accessible to developers.

seMeja API Design Principles

There are some issues that may make it difficult to use traditional API design techniques for the seMeja API. The seMeja API has many incompletely defined variables including shifting requirements and unknown server-side systems. Developers have documented some common-sense principles that can be useful when designing APIs in these circumstances.

These common-sense approaches can be summarized into three design principles:

- The seMeja API should be small. The fewer methods and parameters, the easier it is to be learnt
- The seMeja API should keep options open. Developers may use the API in unforeseen ways, therefore the API needs to be flexible
- The seMeja API should not appear complex. The API should not require the developer to write code that can be handled internally by the libraries

The Design of seMeja API

The seMeja Environment is a three-tier system, as shown in Fig. 5. It sits on top of the Core Operating System and utilizes the system libraries and applications. The tiers are, from bottom to top, the seMeja Services, seMeja API and seMeja Applications.

The seMeja Services layer consists of Core Services and Application Services. As discussed previously, the Core Services provide services used by all applications. The Application Services are domain specific, used by specific applications. In addition to the service components, there are other components that support the services. These include the profile manager, data cache and other possible components.

On top of the seMeja Services is the seMeja API. The API serves as a wrapper, providing standardised access to the various components and agents. The final tier of the seMeja architecture is the applications, which access the seMeja Services through the API layer.

Requirements for seMeja API

In order to design the seMeja API, a study on the requirement of the layer precedes the seMeja API needs to be done, placing a special emphasis on the Core Services. The Core Services can be grouped into two categories: Services that access external functions and services that do not. The services that access external function are called agent services and it includes things like accessing student data, registration information and course material. The services that do not access external functions, or the non-agent services, are based on key technologies that have been identified by the other seMeja research groups.

In term of agent services, there are several common services with respect to the institutions of higher learning. Works that has been done by the following peoples show the importance of the services:

- Course Registration: By (Sherman, 2000; Xue-hua *et al.*, 2012; Dee and Bryan, 2011)
- Course Resources: By (Grabe and Sigler, 2002; Soong *et al.*, 2001; McNaught *et al.*, 1999; Meinel *et al.*, 2002; BI, 2012)
- Assessment: By (Rovai, 2000; Joy *et al.*, 2005; Amelung *et al.*, 2008)
- Research Project Management: By (Li *et al.*, 2007)
- Self administration: By (Pollock, 2003)

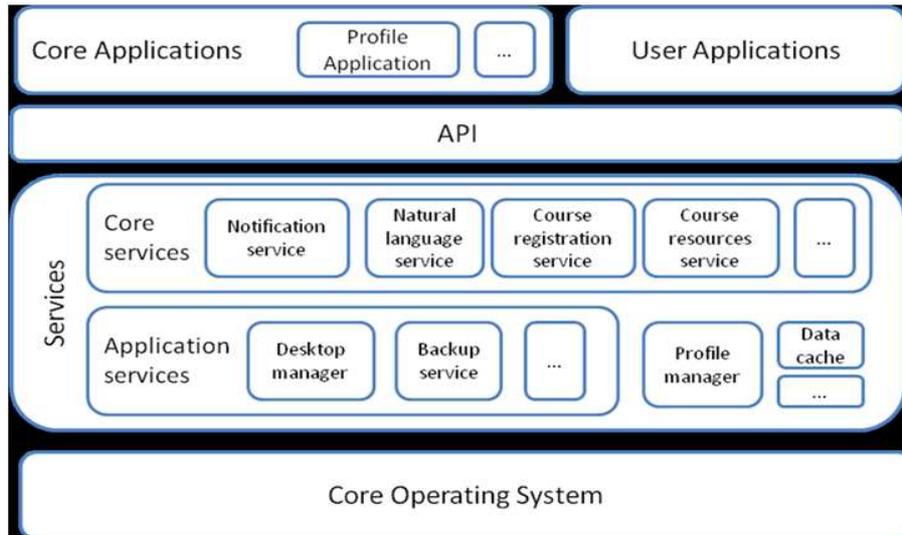


Fig. 5. Architecture of the seMeja environment

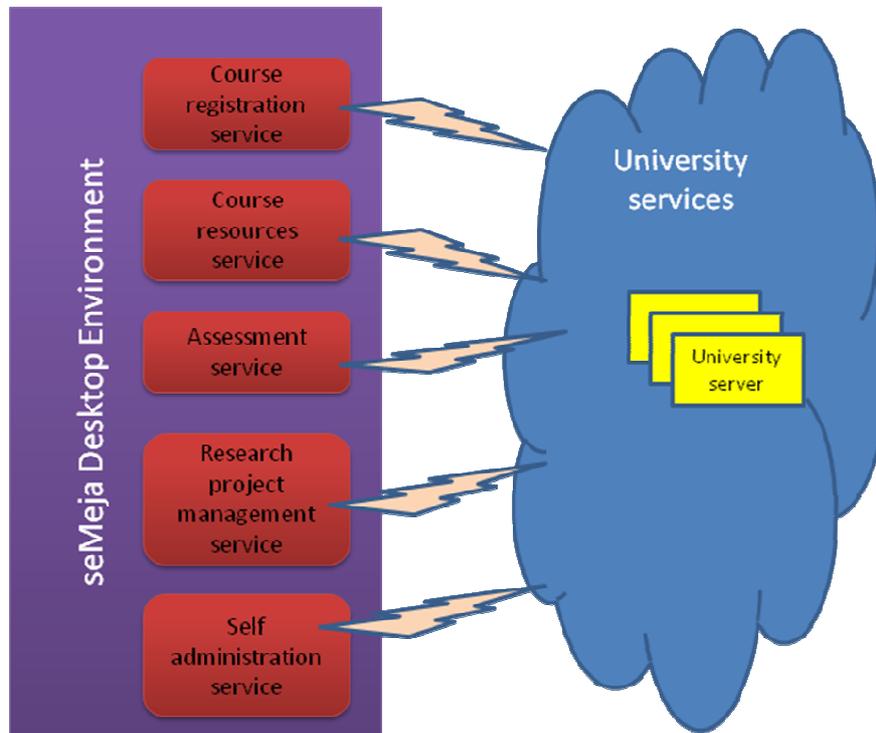


Fig. 6. Agent services in the seMeja desktop environment

Table 2. seMejaAPI agent modules

Function	Description
Course registration	Stores information about the courses that a student is registered for.
Course resources	Handles course materials including lecture notes, manuals, videos and tutorials.
Assessment	Can as simple as a repository of assignment or can support automated assessment.
Research project management	Module used to manage research projects containing interfaces to choose projects and track progress.
Self-administration	Used to view and modify student data.

Figure 6 shows the illustration of the services relationship with the university server, whilst Table 2 describes the agent modules.

In addition to the core agent services, the seMeja Services layer also contains several modules that are not related to university services-either students-centred and/or low barrier for adoption. Hence two other core modules are included: Natural language module and notification modules. Natural language processing would make it easier for students to learn how to use the operating system, consequently, lowering the barrier for adoption. The details of the natural language module are outside the scope of this research and are left open for future research.

Design Concept

CRUD + N Concept

In a traditional API, there would need to be a fixed set of functions and definition for every university services. This would not be practical for the seMeja system, where there is a great deal of variations between the different online systems. The proposed solution is to use the CRUD concepts. CRUD stands for Create, Read, Update and Delete. These principles are familiar to programmers because they are the commonly used for database operations and web services. The CRUD concepts were first used in database management in the 1980s to describe the functions for persistent storage (Martin, 1983). Since the 1980s, these concepts have been applied to many other areas. One of the most widespread applications of CRUD principles is in the Hypertext Transfer Protocol (HTTP) verbs. The HTTP verbs POST, GET, PUT, DELETE map directly to the CRUD principles (NWG, 1999).

The CRUD principles cover most of the functionality required for university services. In addition to the basic four, the seMeja project proposes to add the principle of ‘Notify’. The principle of ‘Notify’ allows applications to register to receive push-style notifications and updates.

The proposed seMeja API uses ideas from REST to provide a simple interface to access university services. The CRUD and notification principles (CRUD+N) are used as method names for Java calls. This means that

there are only 5 methods. The first parameter to each method will be the object that is the target of the method. University objects are similar to REST resources and include things like courses, students, lecturers and documents used as course material. The other parameters contain the rest of the information required to complete the request. Parameter overloading is used to allow different types of data to be used, depending on the object.

Object Description based on Bowlonga Ontology

Based on the CRUD+N principle, the seMeja API has 5 simple methods, with object names to identify the data and overloaded parameters to provide context. This design gives the API the flexibility to access the various university functions identified in the previous section. However, flexibility is useless if there is not enough structure and documentation for a programmer to write codes. Using the CRUD+N principles, the method names for the seMeja API are known. The next step is to determine the objects and parameters used in the methods.

To get an initial set of objects, it is proposed that an existing ontology for universities be used. The Bowlonga Ontology is a good fit for this purpose (Demartini, 2011). The Bowlonga Ontology was created as part of the Bowlonga Process (BBS, 2010). The Bowlonga Process was a multi-national reform process aimed at increasing standardization among universities in Europe. The Bowlonga Ontology defines a set of objects and properties to use in a university environment (Table 3).

By using the CRUD+N principles paired with objects defined in the ontology, a programmer can deduce a set of steps to perform a task. For example, if a programmer wants to create application to allow users to register for courses, the steps would look as described in Table 4.

The same technique of combining CRUD concepts and university objects can be extended to any operation that requires access to university services. These flexible function calls must be combined with detailed documentation complete with examples. The result should be a system that is interoperable between various universities and faculties but still simple and efficient for programmers to use.

Table 3. Examples of Bowlonga ontology

Object	Description
Academic degree	Two levels of academic degree are defined: Bachelor and Master.
Department	A Department can be associated with a Study Track and Teaching Units.
Teaching unit	Several types of teaching units are defined including obligatory, optional and specialisation modules.
Evaluation	Conveys the concept of a grade for a class.
Person	Includes professor and student.
Study program	A program of teaching units and requirements needed to get an academic degree.
Thesis	Scientific work written as part of the academic degree which presents the results of a research project on a topic related to the field of study covering the study program.
Teaching unit	Equivalent to a class.

Table 4. Steps for course registration

Step	Action
1	<i>READ</i> all the teaching units for the computer science department.
2	The application then displays the courses to the student in a nice user interface, allowing the user to pick and choose classes.
3	<i>CREATE</i> a new registration for the Student.
4	Call <i>NOTIFICATION</i> to receive notifications of changes in the teaching unit.

ITALIC indicates CRUD+N principles.
Bold indicates objects from the ontology.

Table 5. seMeja methods and parameters for learning activity object

CRUD+N	CREATE, READ, UPDATE, DELETE, NOTIFY
Ontology Object	Learning_Activity
Description / parameters / sample code	A Learning_Activity is any activity conducted in conjunction with a Teaching_Unit. Examples include labs, assignments and tutorials. Calls to READ retrieve information about a learning activity. CREATE lets a user add new learning activities. For example, a user calls CREATE to submit an assignment. UPDATE allows users to change a learning activity. For example, adding an annotation to a tutorial. DELETE will remove a learning activity. NOTIFY allow users to receives notifications about changes to a learning activity. Possible parameters: Teaching_Unit Example: create("Learning_Activity", "Teaching_Unit=Algorithms101", data);

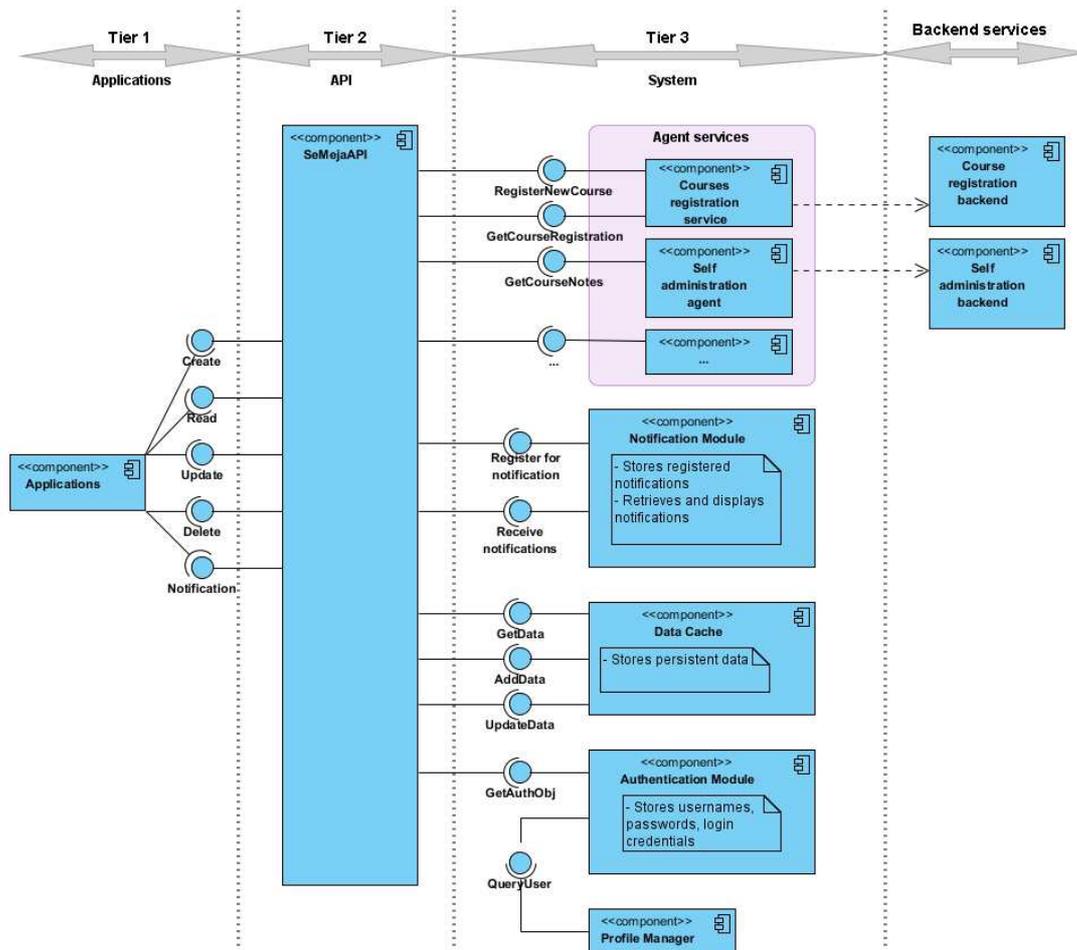


Fig. 7. Components of the seMeja API

Method Summary

Methods

Modifier and Type	Method and Description
int	<code>create(java.lang.String objName, java.util.Map<java.lang.String, java.lang.Object> paramList)</code> Performs a CREATE operation.
int	<code>delete(java.lang.String objName, java.util.Map<java.lang.String, java.lang.Object> paramList)</code> Performs a DELETE operation.
int	<code>notify(java.lang.String objName, java.util.Map<java.lang.String, java.lang.Object> paramList)</code> Performs a NOTIFY operation.
java.lang.Object	<code>read(java.lang.String objName, java.util.Map<java.lang.String, java.lang.Object> paramList)</code> Performs a READ operation.
int	<code>update(java.lang.String objName, java.util.Map<java.lang.String, java.lang.Object> paramList)</code> Performs an UPDATE operation.

Fig. 8. seMeja API method summary

Components of the seMeja API

The seMeja API interacts with other components within the seMeja Operating System (seMeja OS). It also communicates with university services (as shown previously in Fig. 4). These interactions are shown in the component diagram Fig. 7.

The large box in the middle contains the modules that make up the seMeja API. The boxes on the left represent components that are part of the seMeja OS, but not part of the API. Applications interact with the API by calling any of the five CRUD+N methods.

The seMeja API acts as an interface between the applications and the services. For simplicity, Fig. 7 shows only two of the five agent services. All five have similar behaviour. As previously discussed, the agent services access various university services provided by the university servers. Henceforth, the university servers and their functions may be referred to as university server-side services, or simply server-side services. The server-side services are shown as the boxes on the right (Fig. 7).

seMeja API Method

From a developer's point of view, there are only five methods in the seMeja API, each corresponds to the CRUD+N methods. Documentation for the seMeja API was created using Javadoc (Kramer, 1999), which is an industry standard Java documentation technique. Figure 8 shows the Javadoc summary documentation for the 5 CRUD+N methods. Table 5 shows the one example of seMeja API methods and parameters used to develop the prototypes for this research.

Implementation of Application Prototype

Two prototypes were created to showcase that the seMeja API fulfils its requirements. In another work,

the question is can the seMeja API be used to successfully create applications that are useful for university students? The prototypes were chosen to cover a reasonable subset of the CRUD+N calls. The first, a PDF transcript generator, focuses on the READ operation. The second, a Facebook forums tool, shows NOTIFICATION and CREATE operations. Both prototypes are implemented with UKM students in mind, using UKM server-side services.

PDF Transcript Application Prototype

The PDF transcript application is designed as a tool to generate standardised and well-formatted grade transcripts. The PDF format is a widely supported format that is known for ease of viewing and printing. The proposed use of the prototype is two-fold. For students, it provides an easy way to generate a clean grade transcript document than can be printed and/or emailed for internships applications and transfer request. For administrators, a standardised format makes it easier to compare the results of students from different educational programs. The PDF transcript application was chosen as a prototype because it illustrates the use of the READ operation. For many applications, the READ operation is the one that is used the most.

Using CRUD+N and Ontology

By using the CRUD+N principles and studying the Bowlonga ontology, a developer can get a good idea of the seMeja API methods that need to be called. Looking at the information that needs to be displayed, the application programmer are able to deduce that there are 2 READ operations required. The first READ is to retrieve the list of courses and the second READ to retrieve the grade.

By reading the ontology, the relevant objects are identified. For this research, an open source ontology viewer was used to read the Bownlonga ontology more effectively (Mindswap, 2006). In the ontology, courses are labelled as Teaching Units and a grade are labelled as an Evaluation. Using this information, it is possible to work out the steps that the application needs to perform (Table 6).

Components and Processes

Figure 9 shows the components that are used by the PDF transcript prototype. The components are as follows:

- Authentication module

The system already knows who the user is. The authentication module has (or can obtain) the authentication information needed for the other agents

- Course registration agent

The seMeja system interacts with the course registration module to get a list of the user's registered courses

- Student data agent

- This module provides access to student data and is used to obtain student grades

READ Operation for Teaching Units

Each READ operation is a multi-step process. The sequence diagram in Fig. 10 shows the steps for reading the registered courses. First, the application initiates a READ operation, requesting to read a student's registered courses. The registered courses are returned as a list of teaching units. Keep in mind that to perform this process, the student must have logged into the seMeja OS, therefore the seMeja OS already knows who the user is. The system can now query the authentication module. Based on the user and the operation, the authentication module returns the correct authentication object. If the authentication object (for reading registered courses) is not available, the authentication module can then query the user to get the authentication information.

The system now queries the course registration agent, passing it the required authentication object. The course registration agent communicates with the server-side course registration system. The course registration system may take the form of a web page or a database, depending on the university services available.

The course registration agent passes the student's authentication object to the server-side system. The authentication object may contain a user name, a matrix number, a password and other information, as required

by the server-side system. The server-side system passes the requested registration information to the course registration agent. The course registration agent parses the information and converts it into a list of Teaching Units. Finally, the list of registered courses is passed to the application and the application can use the data for further processing. From reading the ontology, it is known that each item in the list of registered courses has a property called has Name. This property is used as identifier. This identifier will be used by the application as a parameter to perform further READ operations.

READ Operation for Grade Information

Once the course registration information has been retrieved, the application needs to get the student's grade for each course. To do that, a further READ operation is needed for each course. The READ operation retrieves the student's grades. For this operation, the server-side student data system needs to know the identity of the student. The external system may also require other authentication information, such as a password, to access private data. Therefore, this operation is similar to the registration information READ operation (Fig. 11).

The READ operation is called, passing in the Teaching_Unit has Name property. First, an authentication object is acquired from the authentication module. This authentication may be the same or different from the previous authentication object, depending on the external agents involved. The authentication object and course identifier are passed to the student data agent. The agent interacts with the external student data system to get the grade. The agent parses the data and converts it to an Evaluation, then returns it to the application. After looping through all the courses, the application now has a complete list of courses, course descriptions and student grades. The application takes that information and uses it to create a standardised PDF file.

Implementation

To retrieve the data, the two service agents (the course registration agent and the self-administration agent) behave like a web browser. The service agent issues a HTTP POST command. The Student Information System website, also known as Sistem Maklumat Pelajar (SMP), returns a web page with the required information. In a web browser, this file is displayed to the user. For the service agents, the relevant data is extracted from the file. The data is packaged into correct objects and returned to the calling application.

Since the actual seMeja Desktop Environment is still in development, the Ubuntu desktop was used to run the prototypes. The first time the application is run, the authentication information is not available, so the authentication module triggers a dialog box to query the user for the information (Fig. 13).

Figure 12 shows a snippet of the PDF transcript code using the seMeja API calls. As discussed, the READ method is used to get a list of registered courses. For each course, the course code is retrieved and passed to a second read operation to get the grade information. After the application runs, a PDF transcript file will be available (Fig. 14). The file can be viewed with a standard PDF viewer or sent as an email attachment.

Table 6. Steps for running the PDF transcript application

Step	Action
1	<i>READ</i> all the teaching units for the student.
2	For each teaching unit, read the evaluation.
3	Create a PDF with the grade information.

ITALIC indicates CRUD+N principles.
Bold indicates objects from the ontology

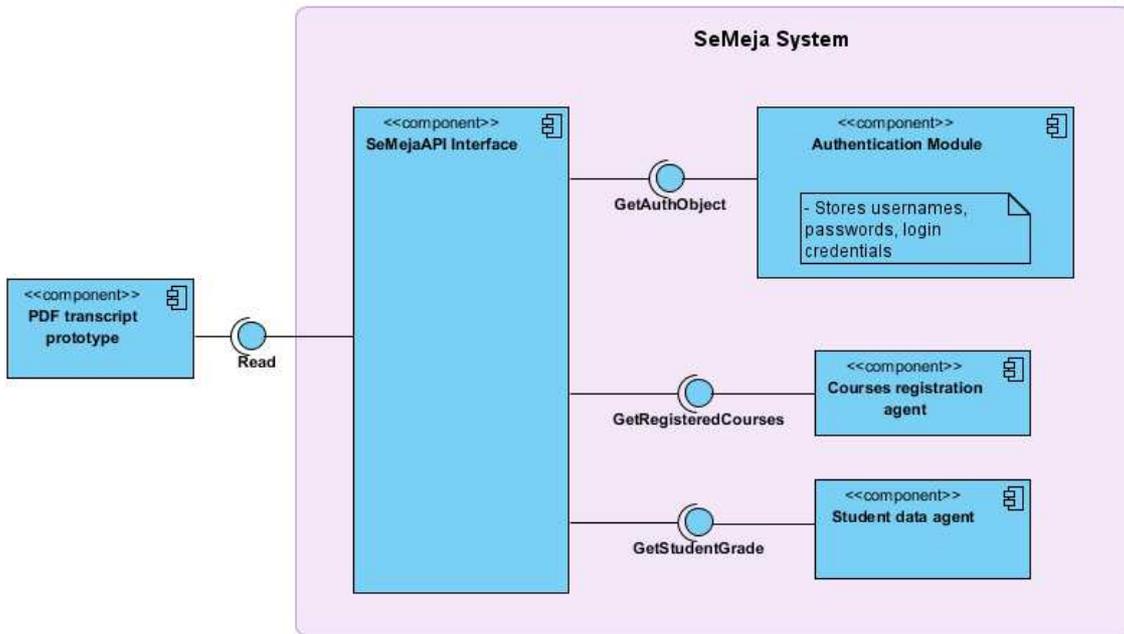


Fig. 9. Component diagram for PDF transcript prototype

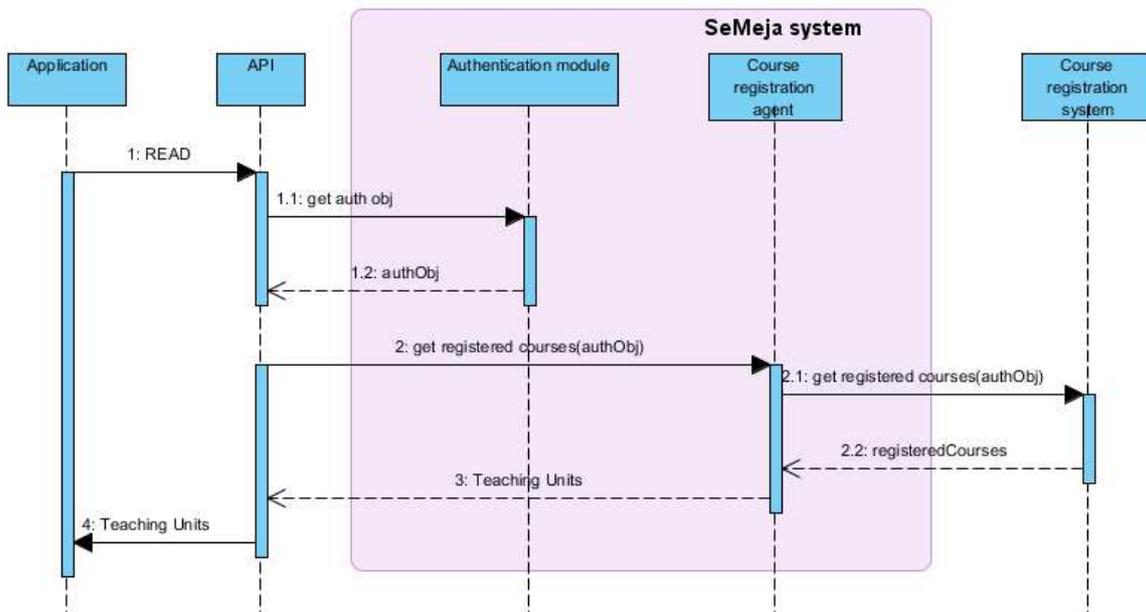


Fig. 10. Sequence diagram for READ teaching unit operation

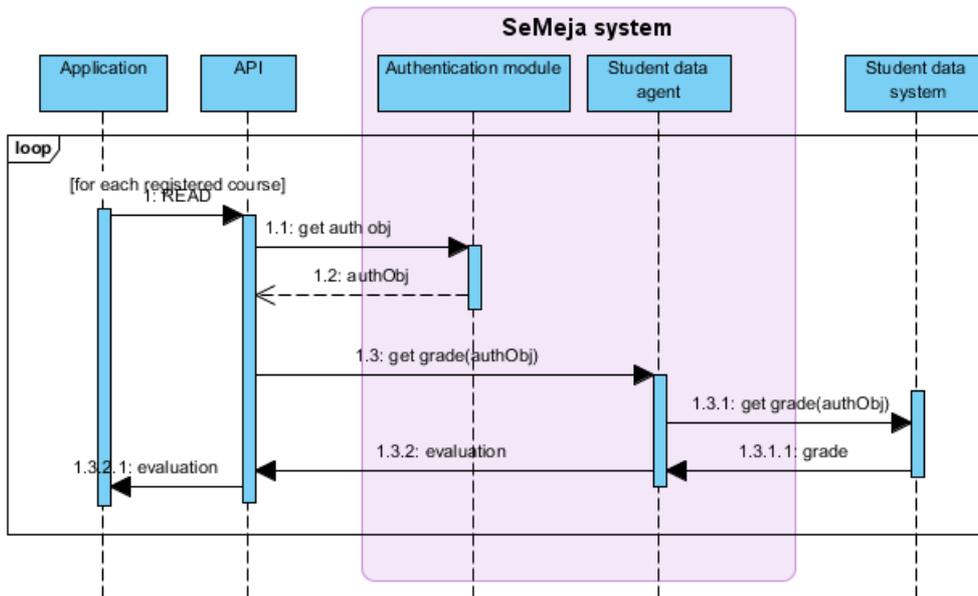


Fig. 11. Sequence diagram for READ evaluation operation

```

try {
    PdfPTable table = new PdfPTable(3);
    addTableHeaders(table);
    java.util.List<CourseInfo> courseInfoList =
        (java.util.List<CourseInfo>) seMejaUKM.read("Teaching_Unit", null);
    for (CourseInfo courseInfo: courseInfoList) {
        Map<String, Object> paramList = new HashMap<String, Object>();
        paramList.put("CourseCode", courseInfo.getCourseCode());
        GradeInfo gradeInfo = (GradeInfo) seMejaUKM.read("Evaluation", paramList);
        table.addCell(courseInfo.getCourseCode());
        table.addCell(courseInfo.getCourseName());
        table.addCell(gradeInfo.getGrade());
    }
    document.add(table);
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
    
```

Fig. 12. PDF transcript code snippet



Fig. 13. PDF Transcript authentication screen capture

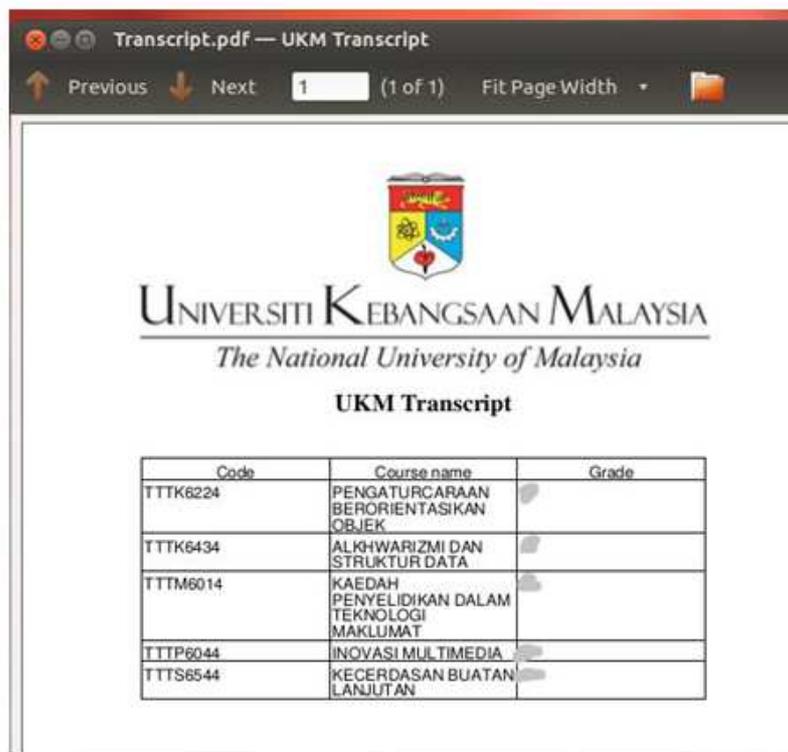


Fig. 14. PDF Transcript screen capture

Facebook Forums Application Prototype

The Facebook forums application prototype is based on a scenario where Facebook is used as a class forum. The class instructor creates a Facebook group and invites students to join the group. The instructor posts announcements, assignments and other course related items to the Facebook group's feed. Students can comment on the posts or create their own posts. The Facebook group provides a forum for students to post questions and discuss topic of interest through the use of comments. This is especially useful at institutions that may not have complete IT infrastructures. However, the approach requires that all students have a Facebook account to participate.

Handling a Facebook account is quite easy for computer literate students. However, it can be challenging for a new user. Facebook has known privacy and security issues. These problems can be managed if the user has a good understanding of internet safety issues and is able to configure Facebook's many privacy options. This degree of internet knowledge may take time for a new user to develop. This may cause problems for new student users as the student will need to access the course forums almost immediately, starting from the beginning of classes.

There are also some institutions that block Facebook on their networks. A part-time student, for example, may access the internet over a public, shared network or through a work network and may find themselves blocked from using Facebook. Therefore, even

experienced student users may still face problems accessing a Facebook based course forum. The purpose of the Facebook forums application is to make the Facebook forum accessible through the seMeja OS. The application allows users to read the forum and create posts without requiring direct access to Facebook and without requiring a Facebook account.

Using CRUD+N Ontology

The steps the application needs to perform can be determined using the CRUD+N concepts along with the Bowlonga ontology. The Facebook forums application needs to do two things. First, it needs to display new messages on the forum. Secondly, it needs to allow the user to post new items to the forum. In order to display the newest messages in near real time, the application needs to call a NOTIFY operation. As previously discussed, calling the NOTIFY will register the application to receive notifications about the Facebook forum. To post new messages to the forum, the application will need to call a CREATE method.

By reading the ontology, the relevant objects can be identified. The Bowlonga ontology shows that a Teaching Unit is related to a Learning Activity. A Learning Activity is something that is done as part of a course. A class forum is a learning activity.

Using this information, it is possible to work out the steps that the application needs to perform, as shown in the Table 7.

Table 7. Steps for running the face book forums application

Step	Action
1	<i>NOTIFY</i> to receive notifications about a learning activity.
2	If a new notification is received, show it to the user
3	On a users' command, <i>CREATE</i> new data for the learning activity (add a new message to the Learning Activity)

ITALIC indicates CRUD+N concepts
Bold indicates objects from the ontology

Components and Processes

The following diagram shows the components that are used by the Facebook forums prototype. The components are as follows:

- Authentication module
The authentication module has the authentication information needed for the various other agents
- Course resources agent
The seMeja system interacts with the course resources module to manage access to the various

NOTIFY Operation for Course Resources

To register to receive notifications from a Facebook forum, the application calls the NOTIFY method, as shown in Fig. 15. When the application calls the NOTIFY method, it passes the name of the Learning Activity as a parameter. When it receives the call, the seMeja API first calls the Authentication Module to get an authentication object. For the Facebook forums, the authentication object is a special token generated by the Facebook system.

Once the seMeja API has obtained the authentication object, the application passes it to the Course Resources service, along with the Learning Activity. Based on the Learning Activity, the Course Resources service can determine which server-side services need to be accessed to register for notifications. In this case, the server-side service is actually a light-weight web application that accesses the Facebook system.

After the notification is executed on the relevant university system, the status of the notification is returned to the notification module. If the registration for notification is successful, the notification module adds the Learning Activity to its list of notifications. The notification list contains the server-side university services that need to be monitored for notifications, along with a list of applications that are registered to receive notifications. Finally, the status of the NOTIFY call is returned to the Facebook forums applications (Fig. 16).

CREATE Operation for Course Resources

The Facebook Forums application has an interface that allows the user to reply to an existing post or create a new post on the forum. To implement this, the application calls the CREATE method. The CREATE sequence is similar to the NOTIFY sequence, as shown in Fig. 17.

The application calls the CREATE method, passing it the name of the Learning Activity, along with the data to be created. In this case, the data is a message that will be posted to the Facebook forum. The seMeja API retrieves the authentication object from the authentication module. The Learning Activity, both the data and authentication object, are passed to the course resources service. The course resources service passes the data and authentication object to the Facebook system. The message is created on the Facebook forum and the status is returned to the application.

Implementation

Accessing the Facebook forums is done through the use of a dummy account. A dummy Facebook account is created specifically for the seMeja system. The dummy account is used to retrieve posts from the Facebook group. It is also used to create new posts. On Facebook, these posts will appear to come from the dummy account. To avoid confusion, the seMeja system will tag the posts with the user's name, which is obtained from the authentication object.

Figure 18 shows a snippet of the Facebook forums code using the seMeja API calls. As discussed, the NOTIFY method is used to register to receive notifications. For this early prototype, this is implemented using a callback handler. More sophisticated message passing methods may be used in later iterations. For this version of the Facebook Forums prototype, the callback simply pops up a message box with the newest message (Fig. 19). Future versions could perform more complicated operations such as message translation, or parsing messages for data for synchronising with calendars or contact lists.

Figure 20 shows a snippet of code using the seMeja API call to create a new post on the Facebook forum. The call is similar to the previous calls. First a dialog box is created to query the user for the message to be posted. Then the message is passed to the seMeja API create function.

As a result of the seMeja API CREATE method, a new message appears on the Facebook forum (Fig. 21). As discussed, a dummy account is used to post the message, so the message is tagged with the user's name.

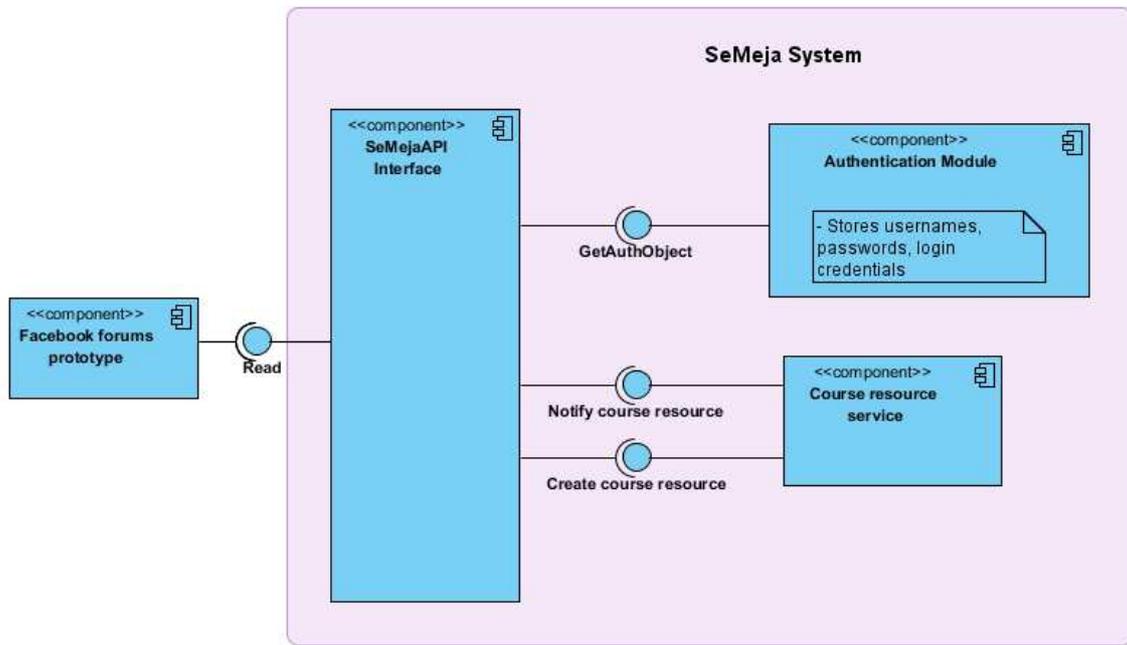


Fig. 15. Component diagram for facebook forums prototype

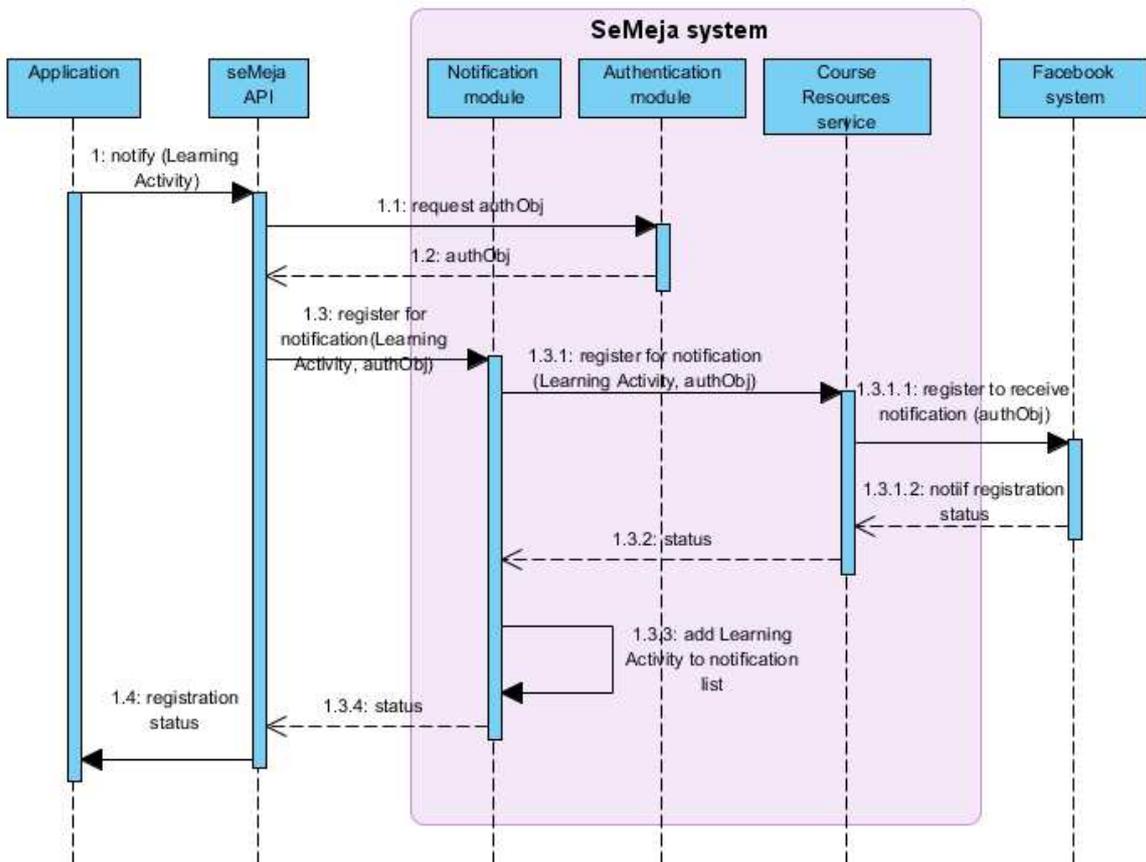


Fig. 16. Sequence diagram for NOTIFY operation

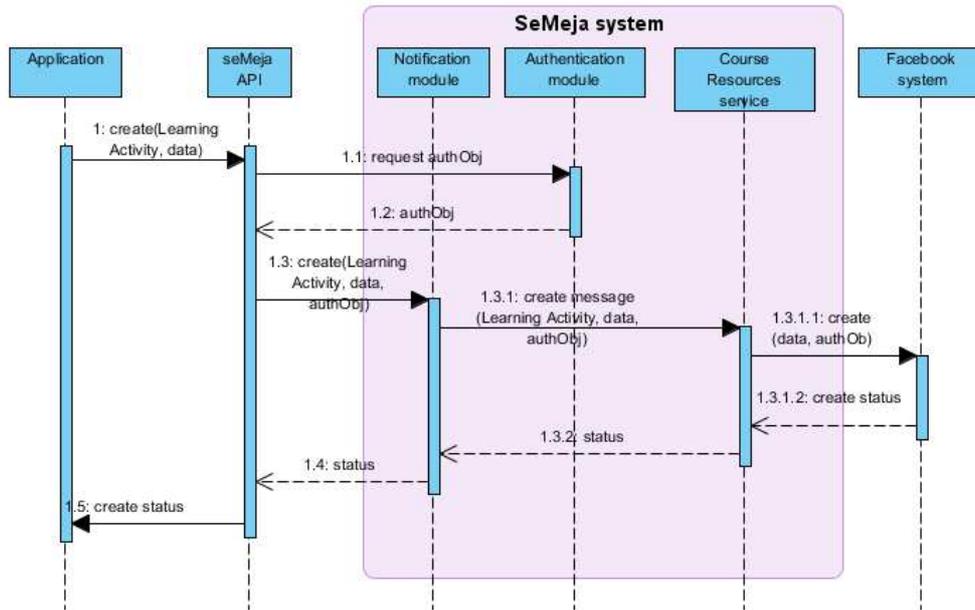


Fig. 17. Sequence diagram for CREATE operation

```

public class FBForumNotificationHandler implements NotificationCallback {
    public void callback(String eventMsg) {
        JOptionPane.showMessageDialog(null, eventMsg,
            "Notification", JOptionPane.PLAIN_MESSAGE);
    }
}

public void registerForNotification() {
    Map<String, Object> paramList = new HashMap<String, Object>();
    paramList.put("NotificationCallback", new FBForumNotificationHandler());
    seMejaUKM.notify("Learning Activity", paramList);
}
    
```

Fig. 18. Facebook forum notify code snippet

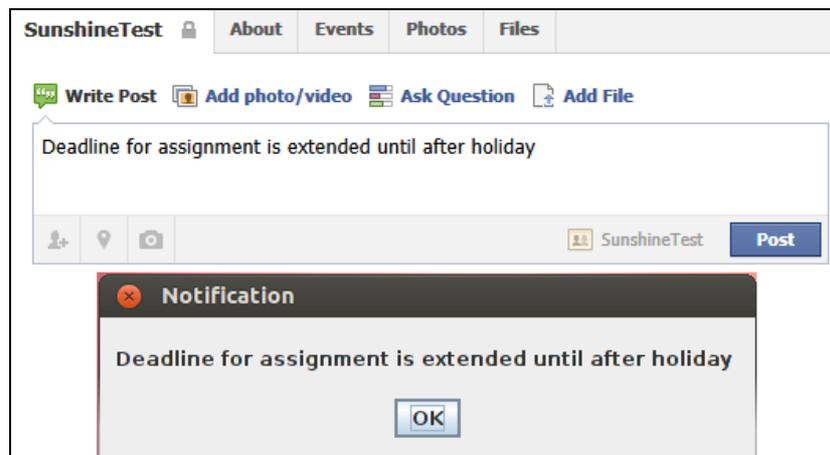


Fig. 19. Facebook forum notification screen capture

```
MessageDialog dlg = new MessageDialog(new JFrame());  
String msg = dlg.getMessage();  
  
Map<String, Object> paramList = new HashMap<String, Object> ();  
paramList.put("Message", msg);  
seMejaUKM.create("Learning_Activity", paramList);
```

Fig. 20. Facebook forum notify code snippet

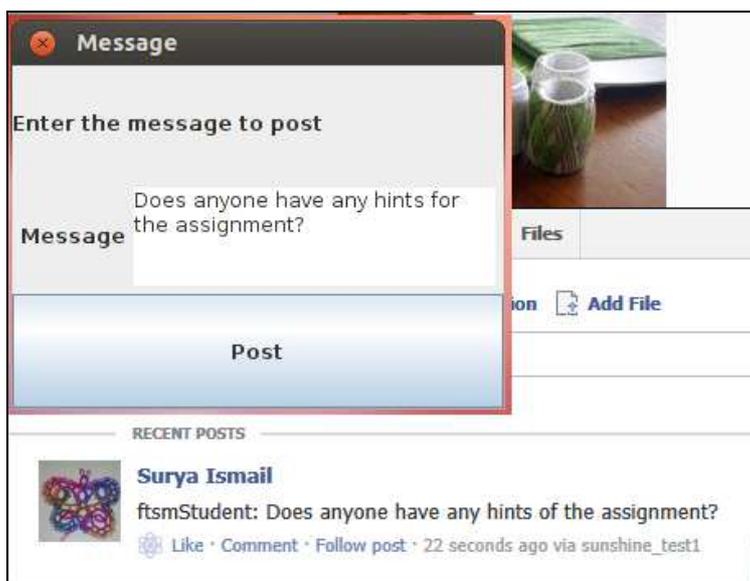


Fig. 21. Facebook forum notification screen capture

Conclusion

The seMeja API succeeds in some ways and fails in others. The API is versatile and flexible. It uses principles and concepts most programmers readily know and hides the intricate details of server-side university services for easy of use. The prototypes work, proving that the concepts behind the seMeja API are useable. It contains some interesting ideas and outlines a way of organising the modules. As an illustration of a design, the seMeja API succeeds. However, at this point, the seMeja API is incomplete. Although the Bowlonga ontology is a good starting point, the documentation and code examples needed by programmers to efficiently write code are incomplete. As a working API that can be used immediately for development, the seMeja API fails.

Future work on the seMeja API can use the work in this research to develop low-fidelity prototypes to create a complete, useful API for the seMeja Desktop Environment.

Funding Information

This work is based on project supported by Universiti Kebangsaan Malaysia under grant UKM-AP-ICT-15-2009.

Author's Contributions

Marini Abu Bakar: Contribute in drafting the article based on Surya's Masters dissertation. Proposed the initial design of the work together with Zarina and Sufian. Supervised Surya's work together with Sufian to ensure it is done correctly according to the objective.

Surya Ismail: Was a Masters student at Universiti Kebangsaan Malaysia. Proposed the detailed design as well as doing the implementation and testing. Written the dissertation on the work. Reviewing the article critically.

Sufian Idris: Proposed the initial design of the work together with Zarina and Marini. Supervised Surya's work together with Marini to ensure it is done correctly according to the objective. Reviewing the article critically.

Zarina Shukur: Contribute in drafting the article based on Surya's Masters dissertation. Proposed the initial design of the work together with Marini and Sufian. Head of the project supported by Universiti Kebangsaan Malaysia under grant UKM-AP-ICT-15-2009.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

- Amelung, M., P. Forbrig and D. Rösner, 2008. Towards generic and flexible web services for e-assessment. Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, (CSE' 08), ACM New York, NY, USA, 13: 219-224. DOI: 10.1145/1384271.1384330
- BBS, 2010. About the bologna process. Benelux Bologna Secretariat.
- BI, 2012. About blackboard. Blackboard International.
- Bloch, J., 2006. How to design a good API and why it matters. Proceeding of the Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages and Applications, (SLA '06), ACM New York, NY, USA, pp: 506-507. DOI: 10.1145/1176617.1176622
- Dee, M.I. and V. Bryan, 2011. First year undergraduate students' perception of the effectiveness and transfer of multimedia training. Proceedings of the World Conference on E-Learning in Corporate, Government, Healthcare and Higher Education, (Oct, 18), Association for the Advancement of Computing in Education, Honolulu, Hawaii, USA, pp: 1884-1889.
- Demartini, G., 2011. The Bowlogna Ontology. Bowlogna.
- Grabe, M. and E. Sigle, 2002. Studying online: Evaluation of an online study environment. *Comput. Education*, 38: 375-383. DOI: 10.1016/S0360-1315(02)00020-9
- Henning, M., 2007. API: Design Matters. *ACM Queue*, 5: 24-36. DOI: 10.1145/1255421.1255422
- Idris, S., M.A. Bakar and Z. Shukur, 2010. Architecture of seMeja desktop system. Proceedings of the International Symposium in Information Technology, Jun, 15-17, IEEE Xplore Press, Kuala Lumpur, pp: 1073-1075. DOI: 10.1109/ITSIM.2010.5561654
- Joy, M., N. Griffiths and R. Boyatt, 2005. The boss online submission and assessment system. *J. Educational Resources Comput.*, 5: 2-2. DOI: 10.1145/1163405.1163407
- Kramer, D., 1999. API documentation from source code comments: A case study of Javadoc. Proceedings of the 17th Annual International Conference on Computer Documentation, Sep. 12-14, New Orleans, Louisiana, USA, pp: 147-153. DOI: 10.1145/318372.318577
- Li, L., P. Li, Q. Liu, J. Zhang and Z. Wang *et al.*, 2007. WebUPMS: A web-based undergraduate project management system. Proceedings of the First IEEE International Symposium on Information Technologies and Applications in Education, Nov. 23-25, IEEE Xplore Press, Kunming, pp: 360-364. DOI: 10.1109/ISITAE.2007.4409304
- Martin, J., 1983. Managing the Data-base Environment. 1st Edn., Sung Kang, pp: 766.
- McNaught, C., J. Kenny, P. Kennedy and R. Lord, 1999. Developing and evaluating a university-wide online distributed learning system: The experience at RMIT university. *Educational Technol. Society*, 2: 70-91.
- Meinel, C., Sack H. and V. Schillings, 2002. Course management in the twinkle of an eye-LCMS: A professional course management system. Proceedings of the 30th Annual ACM SIGUCCS Conference on User Services, (CCS '02), ACM New York, NY, USA, pp: 281-283. DOI: 10.1145/588646.588722
- Mindswap, C., 2006. Swoop-a hypermedia-based featherweight OWL ontology editor.
- MC, 2012. Modular object-oriented dynamic learning environment. Moodle Community.
- NWG, 1999. Hypertext transfer protocol--HTTP/1.1. Network Working Group.
- OLPCF, 2012. One Laptop per Child (OLPC), a low-cost, connected laptop for the world's children's education. OLPC Foundation.
- Pollock, N. 2003. The 'self-service' student: Building enterprise-wide systems into universities. *Prometheus Critical Studies Innovation*, 21: 101-119.
- Robillard, M.P., 2009. What makes apis hard to learn? Answers from developers. Proceedings of the IEEE Software, Nov.-Dec. 2009, IEEE Xplore Press, pp: 27-34. DOI: 10.1109/MS.2009.193
- Rovai, A.P., 2000. Online and traditional assessments: what is the difference? *The Internet Higher Education*, 3: 141-151. DOI: 10.1016/S1096-7516(01)00028-8
- Sherman, J.C., 2000. An experiment in web-based registration for new students in the college of engineering and applied science. Proceedings of the Annual American Society for Engineering Education Conference, 1353: 232-240.
- Soong, M.H.B., H.C. Chan, B.C. Chua and K.F. Loh, 2001. Critical success factors for on-line course resources. *Comput. Education*, 36: 101-120. DOI: 10.1016/S0360-1315(00)00044-0
- SL, 2012. Sugar learning platform. Sugar Labs.
- Xue-hua, L., Z. Zhou-sen, W. Zhen-dong and Y. Xiao, 2012. Research and realization of distributed digital registration system for universities. Proceedings of the International Conference on Computer Science and Electronics Engineering, Mar, 23-25, IEEE Xplore Press, Hangzhou, pp: 490-493. DOI: 10.1109/ICCSEE.2012.56