Middleware to integrate heterogeneous Learning Management Systems and initial results

J.A. Hijar Miranda
Instituto Politécnico Nacional
SEPI-ESCOM
México D.F.

Daniel Vázquez Sánchez
Instituto Politécnico Nacional
SEPI-ESCOM
México D.F.

Dario Emmanuel Vázquez Ceballos
Instituto Politécnico Nacional
SEPI-ESCOM
México D.F.

Erika Hernández Rubio
Instituto Politécnico Nacional
SEPI-ESCOM
México D.F.

Amilcar Meneses Viveros
Departamento de Computación
CINVESTAV-IPN
México D.F.

Elena Fabiola Ruiz Ledezma
Instituto Politécnico Nacional
SEPI-ESCOM
México D.F.

***Abstract*—The use of the Learning Management Systems (LMS) has been increased. It is desirable to access multiple learning objects that are managed by Learning Management Systems. The diversity of LMS allow us to consider them as heterogeneous systems; each ones with their own interface to manage the provided functionality. These interfaces can be Web services or calls to remote objects. The functionalities offered by LMS depend on their user roles. A solution to integrate diverse heterogeneous platforms is based on a middleware architecture. In this paper, a middleware architecture is presented to integrate different Learning Management Systems. Furthermore, an implementation of the proposed middleware is presented. This implementation integrates two different Learning Management Systems, using Web services and XML-RPC protocols to access student-role users capabilities. The result is a transparent layer that provides access to LMS contents.**

***Keywords*—Middleware; Learning Management Systems; Application Program Interface**

I. Introduction

The use of the Learning Management Systems (LMS) has been increased in academic and business communities. These systems are used as auxiliary tools for courses, workshops and training[1] [2]. In [1], the authors refer to the LMS as an emerging technology in education. Among the most popular LMS it can be mentioned Moodle, Blackboard, Claroline, Chamilo, Olat, Sakai, Dokeos, eCollege, Angel and KEWL.

There are diverse LMS, each ones with their particular functionalities [3]. Each LMS define their own user roles. Each role has its own set of features and access methods. The most common user roles are administrator, student and teacher. The administrator manages user accounts and courses, and set permissions for use and gives access to resources of the LMS. A student may enroll in courses; accessing the learning objects associated with these courses. In addition, the student can perform tasks like using forums, chat rooms, video conference or solve exercises and exams. Teachers can update learning objects (such as course materials, videos, etc), enroll students in courses and apply evaluations.

The report [1], also indicates that mobile devices are being adopted in education as means of access to online courses. When trying to access the LMS from a tablet, there are several issues. These issues include: poor usability user interfaces to access unsuitable for tablet; the diversity of mechanisms of interaction via internet; heterogeneity of the features of the LMS, and the type of Application Programming Interface (API) offered.

Some studies suggest using a repository of learning objects that can be accessed by different LMS, such as [4] [5]. Other authors suggest the use of ontologies for handling semantic Web [6]. But the problem of access on mobile devices is not solved. One solution is to use a middleware to integrate a set of basic features of the LMS. This type of solution has been successfully used to solve problems in heterogeneous environments, such as travel agencies, where various services are integrated such as: selling air tickets, car rental and hotel reservations.

In this paper, a middleware architecture is presented to integrate several LMS. This architecture contains components and APIs. This middleware allows a client to connect to various LMS through a software layer. Furthermore, an implementation of the middleware is presented. Web services and protocols based on XML-RPC are used, so that the middleware can interact with various LMS.

This paper is divided in five parts, the first section gives the introductory remarks about LMS and middleware systems, the second part shows the related work, the third part presents the proposed middleware architecture, the fourth part presents a prototype implementing the proposed architecture and the last section discuss the results obtained, the conclusion and future work.

II. Introductory remarks

A. Learning Management Systems

A Learning Management System [7] can be defined as software installed in a server used to manage, distribute and control

distance learning activities of an organization. The main functions of the LMS are: Manage users, resources, materials and learning activities as well as manage access, reporting and manage communication services as discussion forums and video conferences to name a few.

It can be identified two types of LMS[8]: Open Source and Private.

Private LMS are mainly used by companies to manage and keep track of employees through staff training. Advantages of these are: support greater amount of users and courses; more information can be obtained from this platforms; new functionalities and extra reports can be requested. The main disadvantage is that these systems are very expensive. Examples of these platforms are: Blackboard, Desire2Learn, Saba Learning, iLearning, Aulapp, Catedr@, eCollege, Fronter, SidWeb, WebC and WebClass to name a few [3].

Open Source LMS are mainly used at school level to reinforce the basic knowledge as well as keep track of the students. The main advantages of these systems are theme modification and customization of the platform. The main disadvantage of these systems is that they do not have enough documentation or support to make some modifications. Examples of these platforms are: Moodle, Sakai, Chamilo, Docebo, ILIAS, ATutor, Claroline, DaVinci LMS and SWAD to name a few [3].

B. Middleware

A middleware is a distributed programming layer that provides programming abstraction as well as masking of underlying layers such as networks, operating systems, programming languages and hardware. It helps significantly when developing distributed applications. Any middleware works with the differences among operative systems and hardware [9] [10].

Within the general architectural model of a distributed application, a middleware layer uses message-based protocols between processes to provide higher-level abstractions such as remote invocations and events[9]. A middleware provides these features [11]:

- *Location transparency*: A client executing processes is not capable of distinguish if it is executing locally or remote.
- *Communication protocols independence*: The protocols giving support to the abstractions of the middleware are independent of the protocols of underlying transport.
- *Hardware independence*: Components of the distributed system can interact in a proper way independently of the platform executing the process.
- *Operative systems independence*: Abstractions at higher-level provided by the middleware are independent of the underlying operative systems.
- *Use of several programming languages*: Different middleware are designed to allow that distributed applications can be written beyond one programming language. This can be achieved using an Interface Definition Language (IDL).

There are different types of middleware[12] [13] [14]:

- *Message oriented*. Based under the concept of message interception, supports communication between distributed components through message passing. Components can communicate one on one through publication and subscription of data using the global name space. Communication is asynchronous. It is particularly ideal to implement event notification-based as well as publish/subscribe paradigm-based distributed architectures.
- *Object oriented*. Based on Object Broker. Uses the concepts of object oriented programming for the design and implementation of the middleware. Allows independence of each component distribution and the interaction of each component is defined by interfaces. Scalability of this type of middleware is limited.
- *Transaction oriented*. Based on transaction monitoring. Uses the Two-Phases Commit protocol to support distributed transaction. Simplifies the development of a transactional distributed system. However, it causes a unwanted overhead if it is not necessary the use of transactions.
- *Service oriented*. Based on Service Oriented Architecture (SOA), which is a computing paradigm that uses services as main elements to support fast development, low cost and easy composing of distributed applications. A middleware based on this paradigm must show services and manage them through three key components: name service provider, service requester and register. Particularly, provides enough support to service providers so they can show the services and allows to publish their presence in the registry so that the service requesters can find and use the services.

III. Related Work

Nowadays the are some LMS trying to adapt their features towards mobile devices through the development of specific applications for a mobile device platform as well as the use of rich-client architectures to adapt the interface in the mobile device. Some works are mentioned below:

Mobile application developers bound the development of these for one or two platforms given their proliferation due there is no execution support in multiple platforms. A solution is developing web-based applications but there are drawbacks such as adaptability, server overloads and low use of the mobile capacities. A middleware-based architecture can solve the mobile applications requirements through the implementation of rich-client applications in order to manage the differences in heterogeneous platforms [15].

Moodle Mobile project (MM) is an application of Moodle for mobiles based on technologies such as HTML5 that basically is a web client using REST as protocol to obtain and send information to Moodle in the server [16]. The layer is created using HTML5 and CSS3 while interaction with mobiles is provided by Phonegap and uses jQuery for DOM manipulation.

The only feature that all the mobile platforms share is that they have a browser, which is accessible from native code. Each platform allows to instantiate a browser and interact with a Javascript interface from native code[17].

A developer can use different methods to write mobile applications: HTML5, native and cross-platform applications and give solutions to interfaces for different LMS with their own frameworks [8].

The proposal of extending Moodle services [18] towards mobile devices describes a way to integrate mobile devices and educative applications with the LMS Moodle through the use of web services, proposing a set of open specifications of web services to integrate external mobile applications with Moodle.

IV. Middleware architecture

Within this work is presented a proposal of the design of a middleware-based architecture capable of integrate different LMS and manage a set of student role functionalities.

From a functional analysis in different LMS were identified the main student role functionalities which are implemented in all LMS. These functionalities are:

- Student registry.
- List of courses of all LMS connected to the middleware.
- Course enrollment.
- User autentication in the middleware.
- Show student profile.
- Edit student profile.

The middleware encapsulates these functionalities of the LMS reaching an heterogeneity grade among them.

The proposed architecture is shown in figure 1, is weakly-coupled-based and the middle layer is a service oriented middleware. The middleware layer implements the student role functionalities through the interface LMSMiddlewareAPI, and publish a service so the client applications (rich-client) can execute in a remote way these functionalities. The middleware API, together with the methods of DAO component manage the registry, update and obtain data of the users registered in the middleware database. The middleware's LMSAPI component implements methods that manage the access to the LMS as well as functions such as: list courses, course enrollment and obtaining educative contents from the LMS connected to the middleware.

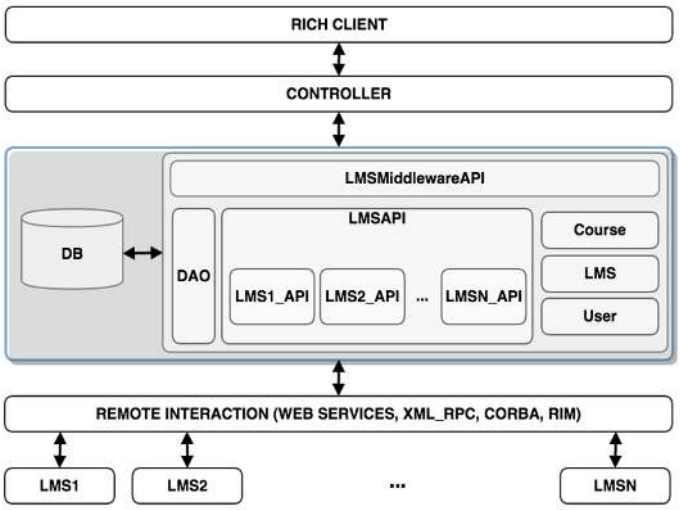

Fig. 1: General design of middleware-based architecture for integrating Learning Management Systems.

A. Rich-client

In the previous architecture it is showed a rich-client layer on the top, which represents web browser clients connected to the middleware through HTTP requests. A rich-client application allows the interface to be adapted dynamically to different devices such as: smartphones, tablets, laptops or desktop computers. Besides, is independent of the device platform, hardware or operative system. Therefore a rich-client application allows the management of different platforms and provides support to show the information of the LMS in a proper way. Together with a middleware-based architecture is possible to create an open system. The rich-client has different modules that a student-role user can use to access data from different LMS. These modules are:

- *Login*. Grants access to the system. This module validates the user in the middleware database.
- *User registry*. Adds a user in the middleware database. Requested fields are: Name, last name, user name, password, city, country and e-mail. These data are essential to register a student in the LMS.
- *List of courses*. Show the courses of the LMS connected to the middleware.
- *Show user profile*. Show the current data of the user.
- *Edit user profile*. Show an application form to modify the information of the user in the middleware database. If the user is registered in one or more LMS the information is updated in each LMS database.
- *Course Detail*. Show the contents of the selected course. In this module is presented the option of course enrollment. The middleware database contains the information of users subscribed to the course. For users enrolled in the course it is showed a leyend that says "you actually are registered to this course".

B. Controller

The rich-client controller component allows the interaction between the middleware and the rich-client layers. It encapsulates functions that make dynamically the rich-client presentation. At this component level it must be programmed the invocations towards middleware functions. To communicate with the middleware layer, the component must: know the location of the middleware (this is reached parameterizing important data such as: IP, port and name service); having implemented a

communication protocol to call middleware remote procedures (RPC, XML-RPC, RMI).

C. LMSMiddlewareAPI

Every middleware in any distributed system needs a programming interface (API) that allows the communication with the upper and lower layers of this component. According with [19], a proper API design for the application must be small and sufficient, it means, it must not implement unnecessary functions. Also, the methods names must describe clearly the function they accomplish. Following these statutes, we have designed a proper API for the middleware of our arquitecture which allows the communication with the rich-client. The design of this API is based on the functional analysis previously mentioned where there were presented the main functionalities that were looking for encapsulate. The methods that conform this API are shown below:

- *lms_validateAndObtainUser*: Validates the existence of the user from a given user name and password, if it exists the method returns the user.
  Parameters:
  String : username
  String : password
- *lms_registerUser*: Registers a user in the database, if it exists returns the user id.
  Parameters:
  Associative Array : new_user
- *lms_updateUser*: Updates the information of the user in the database without considering the username. If the user exist in the LMS the information is updated in the LMS as well.
  Parameters:
  Associative Array : user
- *lms_obtainUserByUsername*: Obtains a user from the database given a username.
  Parameters:
  String : username
- *lms_listCourses*: List all the courses contained in all LMS connected to the middleware. If there is not a LMS connected to the middleware a void list is returned.
  Parameters:
  None
- *lms_obtainCompleteCourse*: Obtain the general contents of the course from the LMS.
  Parameters:
  Associative Array : Course
- *lms_isRegisteredToACourse*: Verifies if the user is registered in a given course.
  Parameters:
  int : idUser
  Associative Array : Course
- *lms_subscribeToCourse*: Subscribe the user in the course. Indeed adds a registry of the subscription in the middleware database. If the user is not enrolled in the LMS course, it makes the enrollment in the LMS course.
  Parameters:
  Associative Array : user
  Associative Array : Course

To try to standardize the data type and can be used independently of the communication protocol, there were used native data types for object oriented languages. In some functions, is used a data type named Associative Array, that use named keys assigned to a value. This array can be used in many object oriented languages, for example in PHP, or in Java this arrays are known as Map data type.

The middleware must publish these methods as a service so they can be accessed. The rich-client controller can communicate with the middleware invoking the methods of this API through a remote communication protocol such as RPC, RMI, XML-RPC and SOAP to name a few.

D. Middleware Database

The database in the middleware layer store information linked to the users that access through the rich-client as well as information of the LMS located in the lowest layer of the architecture. Besides it is stored information that involves users with courses. The communication with the database is achieved through the DAO layer within the middleware which is invoked directly from the methods of the LMSMiddlewareAPI component. The methods of DAO allow the update and insertion of user data and information linked to the courses.

E. LMSAPI

Although there is an API for communicating the upper layers of the middleware (towards rich-client) there must exist an API in charge of communicate the lower layers (towards LMS). LMSAPI function is to communicate the middleware with the LMS platforms, implementing necessary methods that invoke the required functions of these platforms. Due to the existence of the LMS heterogeneity as showed in the figure, it must exist a dedicated component that implements the methods calls for each LMS. For this, it can be useful the public APIs of the LMS in the case there is documentation of the APIs. The programming can change depending of the way the LMS publish their services, the type of communication required and the data type requested in the input and output parameters. Now is presented the methods of the LMSAPI component which must be implemented in each LMS*N*_API:

- *registerUser*.
- *updateUser*.
- *obtainUser*.
- *listCourses*.
- *courseEnrollment*.
- *obtainCompleteCourse*.

These methods represent the basic functionalities of the LMS for student-role users and are invoked from the methods of the LMSMiddlewareAPI component. These functionalities invoke propietary methods of the APIs from the LMS.

F. Types of objects

With the purpose of a better information management in the application programming, the middleware layer can encapsulate own data of entities in objects as data types. These data types are *LMS*, *User* and *Course*.

The User data are used to encapsulate information provided from the rich-client and from LMS, to store them into the middleware database. The LMS data was used to encapsulate information provided from the database and are used in methods that access the LMS. The Course data are used to encapsulate information provided by the LMS and send them to the rich-client.

The LMS attributes are in general: id; name/type; url; username/password (administrator); web service name, token/secret key.

The attributes of a course are: id; idLMS; category id; course name; course short name; description; contents and start date.

V. Prototype

The prototype presented in this work consists in the development of an application within the middleware-based architecture presented in the previous section. The specific architecture for this prototype is shown in figure 2.

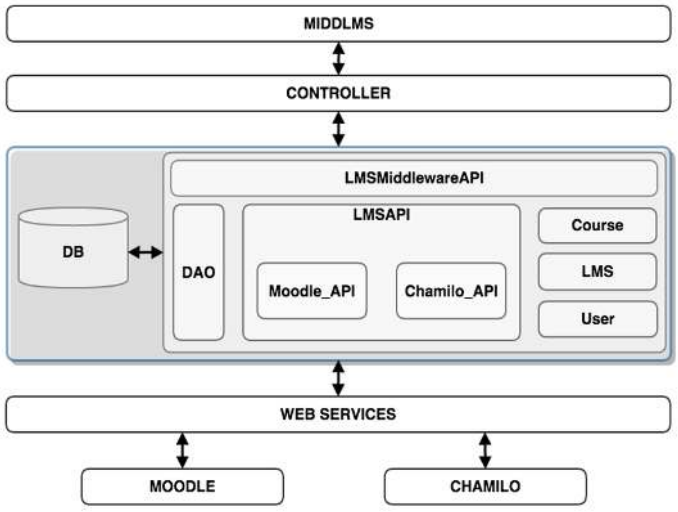

Fig. 2: Specific architecture for prototype application.

The prototype was developed with HTML5 and PHP for the rich-client layer and PHP language was used for developing the middleware layer. The type of communication between rich-client and the middleware is through XML-RPC protocol, used to achieve remote communications and create web services. It is important to mention that for operation of the XML-RPC for PHP it is necessary to have three libraries: xmlrpc.inc, xmlrpcs.inc and xmlrpc_wrappers.inc. The rich-client must know the location of the published services by the middleware. This information is provided by the configuration file conf.php of the rich-client. Also, the middleware must know the location of the LMS. This information is stored in the middleware database.

The LMS used to verify the middleware functionality in this prototype were Moodle and Chamilo. As shown in figure 2, the remote interaction between the middleware and the LMS layers is through web services. Moodle and Chamilo were installed in different servers with the purpose to prove the transparency at location level. Analogous, the middleware was set in a different server.

The middleware needs to know the location of the LMS in order to obtain the information of these. Also, it is necessary to authenticate in a remote way for this to be possible. During the development of this prototype it was not possible to find a standard way to access different LMS. The information of authentication used for Moodle and Chamilo differs and must be stored in the database for being used by the middleware. Here is presented the authentication way to access contents of both Moodle and Chamilo.

A. MoodleAPI

In order to access the functionalities in Moodle is necessary to have a token which is a key that grants permissions to a user to access and use the functionalities of the web services. Moodle has implemented functions to obtain a token. In this way, the middleware can access the functions of Moodle. To achieve this in real time, in Moodle were created a user with permissions to generate tokens from a HTTP calls which must contain as parameters the username and password of the user who has the permissions to generate the token. This information is important to the middleware so it must be stored in the database. With the token, the middleware calls the functions of Moodle through public web services using REST, concatenating the token with the name of the required function. The functions of the web services used were:

- *core_user_create_users*. This function allows the creation of one or more users in Moodle.
- *core_course_get_courses*. This function obtain a list of courses in Moodle.
- *core_user_get_users_by_field*. This function obtain user data given a username.
- *core_course_get_contents*. This function obtains the contents of one or more courses available in Moodle.
- *enrol_manual_enrol_users*. This function enrolls one or more users in a given course.

B. ChamiloAPI

Chamilo web service uses SOAP as communication protocol so it is necessary to create a client that communicates with Chamilo through SOAP objects. To achieve the connection it is necessary to know the secret_key of Chamilo, which is a string of encrypted characters that grants access to the web service methods. The secret_key is part of the parameters in the SOAP calls for Chamilo. This secret_key can not be generated in real time as the token in Moodle, exist within the configuration file config.php in the file system of Chamilo and can be modified by the administrator of Chamilo. For this prototype the secret_key of the installation of Chamilo was used in the middleware to access the functions. The functions of the web service of Chamilo are described below:

- *WSCreateUserPasswordCrypted*. This function creates a user but the password must be encrypted with sha1 method.
- *WSCMUser.find_id_user*. This function returns the user data given a username and password.
- *WSListCourses*. This function search and list all the courses available in Chamilo.
- *WSCourse.SubscribeUserToCourse*. This function enrolls a user into a given course available in Chamilo.
- *WSCourse.GetCourseDescriptions*. This function obtains the description of a specific course available in Chamilo.

VI. Conclusion

From the prototype presented, it is suggested that the design of a middleware-based architecture is a factible option to achieve the integration of different LMS platforms and create a cross-platform system between LMS. In this architecture is proposed that the encapsulated functionalities of the LMS in the middleware layer must be used as services which are available anytime as well as consumed by the clients when required. It means that the information is obtained under demand. For this reason the middleware has a weakly-coupled architecture. This type of architecture is used when modules or layers of a system are independent among them and interact when it is necessary. Weakly-coupled architectures are used in service oriented systems. This middleware encapsulates functions and publish them as services, so the middleware is service oriented. Besides, encapsulates the general information provided by the LMS and the rich-client in object types that are used to manage data, beyond learning objects. Internally, the middleware works under an object oriented scheme.

Also some LMS publish their services to access contents in a remote way. To use this services, there is an authentication way that varies depending on the LMS. In the prototype, the objective was to integrate Moodle and Chamilo. Web services of Moodle and Chamilo were used and both differ in their access way, as an example Moodle uses tokens generated by users while Chamilo uses a static private key. The middleware is in charge of hide the access types to the users that obtain the contents of the LMS through the rich-client. More over not all Open source LMS have web services to access remotely. Open source LMS have an API that is used locally to deploy their contents. A strategy to incorporate them into the architecture is to create an associated web service to the middleware that implements a set of methods which invokes the local API functions of the LMS. The web service serves as a link between the middleware and the LMS. The prototype uses this strategy implementing the XML-RPC protocol.

As future work, the above strategy might be implemented, besides supplementing the current middleware. In the prototype there were used only two Open source LMS, Moodle and Chamilo. To supplement the middleware functionality it is possible to extend the prototype implementation with the LMS, not only for open source but privates too. In the case of private LMS it must be identified if it counts with a web service that can be useful to link it with the middleware. Otherwise, it must be explored if it is a factible strategy to create a web service for this type of LMS.

Acknowledgment

The authors would like to thank Instituto Politecnico Nacional, SIP-IPN, SEPI-ESCOM, CINVESTAV and COMECyT.

References

[1] L. Johnson, S. Adams Becker, M. Cummins, A. Estrada, V.and Freeman, and H. Ludgate, "Nmc horizon report: 2013 higher education edition," The New Media Consortium, Tech. Rep., 2013.

[2] S. Kurkovsky, "Integrating mobile culture into computing education," in *Integrated STEM Education Conference (ISEC), 2012 IEEE 2nd*. IEEE, 2012, pp. 1–4.

[3] D. Vazquez Sanchez, E. H. Rubio, E. F. Ruiz Ledesma, and A. M. Viveros, "Student role functionalities towards learning management systems as open platforms through mobile devices," in *Electronics, Communications and Computers (CONIELECOMP), 2014 International Conference on*. IEEE, 2014, pp. 41–46.

[4] B. Simon, D. Massart, F. Van Assche, S. Ternier, E. Duval, S. Brantner, D. Olmedilla, and Z. Miklós, "A simple query interface for interoperable learning repositories," in *Proceedings of the 1st Workshop on Interoperability of Web-based Educational Systems*, 2005, pp. 11–18.

[5] M. G. Nascimento, L. O. Brandao, and A. A. Brandao, "A model to support a learning object repository for web-based courses," in *Frontiers in Education Conference, 2013 IEEE*. IEEE, 2013, pp. 548–552.

[6] P. Raju and V. Ahmed, "Enabling technologies for developing next-generation learning object repository for construction," *Automation in Construction*, vol. 22, pp. 247–257, 2012.

[7] M. Szabo, "Cmi theory and practice: Historical roots of learning managment systems," in *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, vol. 2002, no. 1, 2002, pp. 929–936.

[8] S. Watermeyer, "Extending sakai web services for mobile application support."

[9] C. George, D. Jean, and K. Tim, "Sistemas distribuidos, conceptos y diseño," *Addison Wesley*, 2007.

[10] M. Van Steen, "Distributed systems principles and paradigms," *Network*, vol. 2, p. 28, 2002.

[11] C. Britton and P. Bye, *IT architectures and middleware: strategies for building large, integrated systems*. Pearson Education, 2004.

[12] L. Qilin and Z. Mintian, "The state of the art in middleware," in *Information Technology and Applications (IFITA), 2010 International Forum on*, vol. 1. IEEE, 2010, pp. 83–85.

[13] V. Issarny, M. Caporuscio, and N. Georgantas, "A perspective on the future of middleware-based software engineering," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 244–258.

[14] L. Jingyong, Z. Yong, C. Yong, and Z. Lichen, "Middleware-based distributed systems software process," in *Proceedings of the 2009 International Conference on Hybrid Information Technology*. ACM, 2009, pp. 345–348.

[15] I. M. T. Hernandez, A. M. Viveros, and E. H. Rubio, "Analysis for the design of open applications on mobile devices," in *Electronics, Communications and Computing (CONIELECOMP), 2013 International Conference on*. IEEE, 2013, pp. 126–131.

[16] (2014, Sep). [Online]. Available: http://docs.moodle.org/dev/Moodle_Mobile

[17] A. Charland and B. Leroux, "Mobile application development: web vs. native," *Communications of the ACM*, vol. 54, no. 5, pp. 49–53, 2011.

[18] M. J. Casany, M. Alier, E. Mayol, J. Piguillem, N. Galanis, F. J. García-Peñalvo, and M. A. Conde, "Extending moodle services to mobile devices: the moodbile project," in *UBICOMM 2012, The Sixth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2012, pp. 24–28.

[19] D. Jacobson, D. Woods, and G. Brail, *APIs: A strategy guide*. " O'Reilly Media, Inc.", 2011.