Navarro, A., Cigarrán, J., Huertas, F., Rodríguez-Artacho, M., & Cogolludo, A. (2014). An Integration Architecture of Virtual Campuses with External e-Learning Tools. *Educational Technology & Society*, *17* (3), 252–266.

An Integration Architecture of Virtual Campuses with External e-Learning Tools

Antonio Navarro[1*], Juan Cigarrán[2], Francisco Huertas[3], Miguel Rodríguez-Artacho[2] and Alberto Cogolludo[2]

[1]Departamento de Ingeniería del Software e Inteligencia Artificial, Facultad de Informática Universidad Complutense de Madrid, C/ Profesor José García Santesmases s/n, 28040 Madrid, Spain // [2]Departamento de Lenguajes y Sistemas Informáticos, Facultad de Informática Universidad Nacional de Educación a Distancia, C/ Juan del Rosal 16, 28040 Madrid, Spain // [3]Center for Open Middleware, Universidad Politécnica de Madrid, CTB building, floor1, Campus de Montegancedo, 28223 Pozuelo de Alarcón, Spain // anavarro@fdi.ucm.es // juanci@lsi.uned.es // francisco.huertas@centeropenmiddleware.com // miguel@lsi.uned.es // acogolludo@lsi.uned.es

[*]Corresponding author

(Submitted July 29, 2013; Revised October 18, 2013; Accepted December 20, 2013)

ABSTRACT

Technology enhanced learning relies on a variety of software architectures and platforms to provide different kinds of management service and enhanced instructional interaction. As e-learning support has become more complex, there is a need for virtual campuses that combine learning management systems with the services demanded by educational institutions and users. However, nowadays, the functions of virtual campuses need to be enhanced with external e-learning applications. This paper deals with the issue of integrating these applications in virtual campuses using a software architecture based on the merging of mashup and multitier patterns. Our solution is based on an asymmetric architecture that provides seamless interconnection of the external tools with virtual campuses. We illustrate it with a real example of the integration of a research prototype that searches for and retrieves learning objects with a virtual campus designed independently of the underlying e-learning platform.

Keywords

Software architecture, Software patterns, LMS, Moodle, Sakai

Introduction

Looking back over the past decade we have witnessed the rapid evolution of technology-enhanced learning, leading to the appearance of a heterogeneous set of e-learning tools, including Learning Management Systems (LMSs), virtual campuses, Managed Learning Environments (MLE), virtual object editors, Massive Open Online Courses (MOOCs) and serious games. This is only a small subset of the e-learning tools that can be found in educational institutions.

However, a constant in educational institutions is the need to manage courses, students and teachers, a need usually addressed by virtual campuses (CCP, 2012). According to Yábar et al., (2007), virtual campuses play several roles: they provide support for face-to-face teaching, encourage teaching innovation and the authoring of learning material, foster communication between the various different users, facilitate learner monitoring, self-learning and self-assessment, and provide blended teaching experiences with different degrees of virtuality.

These campuses are software applications that can be conceived from different points of view. According to Van Dusen (1997), virtual campuses are a metaphor for the electronic teaching, learning and research environment created by the convergence of several relatively new technologies including, the internet, World Wide Web, computer-mediated communication, video conferencing, multimedia, groupware, video-on-demand, desktop publishing, intelligent tutoring systems, and virtual reality. In more recent studies (Allison & DeBlois, 2008; Epper & Garn, 2004; Green, 2005; PLS, 2004) virtual campuses are understood, in a broader sense, to be the integration of Information and Communication Technologies (ICT) in terms of both educational and organizational university settings. This matches our view of virtual campuses as modular software systems used by higher education institutions to give support to their teaching and learning activities (Navarro et al., 2012a).

As a first step, virtual campuses were built using a simple LMS for dealing with core e-learning activities. Educational institutions quickly realized that an LMS was not enough to deal with the management activities (i.e., registration in the LMS course of a group of students enrolled in an in-person course) involved in a virtual campus. So, virtual campuses evolved to become complex applications. The virtual campus deployed at the *Universidad Complutense de Madrid* (UCM) is a complex software system made up of four different applications (Navarro et al., 2012a): (i) two LMSs (Moodle 1.9 & Sakai 2.4), responsible for core e-learning facilities (e.g., content publication); (ii) a J2EE (Weaver, 2004) data load application, responsible for transferring course and user data from the university management system to the virtual campus administration database; (iii) a J2EE administration application, responsible for administrative tasks (e.g., changing a password); and (iv) an integration page that provides a unified view of the courses virtualized in different LMSs.

However, once virtual campuses were running, users demanded three different types of feature (Navarro et al., 2010): (i) the use of different LMSs in the same virtual campus; (ii) independence of the underlying LMS; and (iii) the inclusion of additional e-learning features not contemplated in LMSs. In other papers we have focused on solving issues (i) and (ii). This paper focuses on the third issue.

We have found in the literature two main approaches for dealing with the integration of software applications: (i) those based on service integration using a *multitier architecture* (Alur et al., 2003; Fowler, 2002) such as *Event-Driven Architecture*, EDA (Etzion & Niblett, 2010), *Service Oriented Architecture*, SOA (Erl, 2005) and classic integration patterns (Juric et al., 2001); and (ii) those based on mashup integration (Hanson, 2009; Ogrinz, 2009).

The first type focuses on service integration and does not usually consider cases where one application needs to use the services of another through its user interface. These integration patterns thus focus on isolating service invocation, monitoring transactions and managing interchanged data, omitting user interface integration. The second type, based on mashups, basically pick up information gathered from different service providers and display it in a user-friendly way. Interaction between the different service providers can exist, but this is limited. In addition, the portal (i.e., the result of the mashup) is a new application that probably did not exist previously (Adams & Gerken, 2008; Ogrinz, 2009).

When integrating virtual campuses with external e-learning tools we have considered both types of integration: service-based and user-interface based. Thus, the external tools have to invoke virtual campus services, for example, to upload content for a specific course. In addition, because external e-learning tools usually have a complex user interface, the virtual campus has to render the external tool user interface in the context of its own user interface, enabling the integrated invocation of the external tool services. For example, virtual campus users may need to visualize specific content, previously generated and uploaded in the virtual campus, using the external tool user interface.

The integration of external e-learning tools in the context of virtual campuses has the following integration requirements: (i) two standalone applications must be integrated: the virtual campus and the external e-learning tool; (ii) the virtual campus contains key information about users and courses; (iii) the external tool is able to find or create contents that have to be deployed (or uploaded) to specific courses for particular users of the virtual campus; (iv) because of the preceding requirement, the external tool has to be invoked for a user who is logged on in the virtual campus, thus providing a context for the content to be uploaded; and (v) the external tool has a complex user interface that makes its integration in the virtual campus user interface necessary.

Taking these requirements into account, we can see that these applications play two different roles, which can be generalized for other contexts: on the one hand, the virtual campus is the *host* application, whose user interface has to embed the external tool user interface, providing a context for the uploading of content provided by the tool. On the other hand, the external tool is the *guest* application, which has to be able to respond to virtual campus requests via its integrated user interface (the terms “host” and “guest” are ad-hoc terms defined in this paper).

This raises an important research question: how to provide integration architecture for independent software applications that share services through an embedded user interface, based on the use of design patterns. The use of design patterns guarantees maintainability and other positive characteristics (enumerated in the related work section) in software architectures. The research question arose in the context of the AACV Project, where we had to integrate a virtual campus that we have created with an external e-learning tool.

Reviewing the literature about software integration, we have found that solutions for service integration do not pay attention to user interface integration and those focusing on mashup definitions do not pay attention to complex service interactions between both types of application. This paper provides a generic software architecture for the integration of virtual campuses with external e-learning tools, combining mashup and multitier architecture patterns. The proposed architecture is contextualized in terms of e-learning applications, but could be reused in other applications with similar requirements. Section 2 presents related work. Section 3 describes the proposed integration architecture. Section 4 provides an example of this architecture, describing the integration between a virtual campus and an external e-learning tool, going more deeply into details than the proof of concept architecture described in (Navarro et al., 2012b). Section 5 provides educators' with a simple description of the overall approach described in this paper. Finally, Section 6 describes conclusions and future work.

Related work

Currently almost any entity of medium size makes extensive use of ICT. The bigger the entity, the more likely it is to need to use different software systems to manage different aspects of its business model: payroll, sales management, supplier management, business intelligence, etc. Although some years ago the main problem was the development of these systems, in recent years the main problem has become the integration of the software systems deployed in an organization (Alur et al., 2003; Fowler, 2002). Starting with classic integration patterns (Juric et al., 2001) several software architectures have been proposed to ease the task of integrating different systems that can run as stand-alone. Two of these architectures are *Service-Oriented Architecture* (SOA) (Erl, 2005), and *Event-Driven Architecture* (EDA) (Etzion & Niblett, 2010). SOA architecture is based on the concept of web service: A software function synchronously invoked over a network, irrespective of the platform on which this service is implemented. EDA architecture is based on the concept of an event: a stimulus able to initiate an asynchronous service invocation.

Although SOA and EDA architectures have utility *per se*, they show their full potential in the context of *multitier architecture* (Alur et al., 2003). This architecture divides the software system into five tiers:

- *User tier*: The system actors that interact with the software application.
- *Presentation tier*: Responsible for displaying the application's functions.
- *Business tier*: Responsible for the implementation of the business rules.
- *Integration tier*: Responsible for interaction with the resource tier.
- *Resource tier*: The resources used for the software applications, such as databases or other systems.

Multitier architecture is a de facto standard in the development of enterprise applications (Fowler 2002), because it can easily accommodate SOA or EDA solutions using *web service broker* and *service activator* patterns (Alur et al, 2003).

Mashup literature provides another source of integration patterns (Hanson, 2009; Ogrinz, 2009). A mashup is a new application genus that describes the combination of two or more sources into an integrated site, the output usually being created by one of the sites participating in the mashup (Ogrinz, 2009). Mashups have evolved into complex mixtures of applications, and mashup patterns have been defined to deal with this complexity. These patterns can be classified as follows (Ogrinz, 2009):

- *Harvest patterns*. A general class of solutions that lend themselves to obtaining data from sources outside the reach of traditional tools.
- *Enhance category*. This demonstrates how systems can be extended and improved.
- *Assemble patterns*. These show how fresh solutions can be minted by combining data and presentation from multiple sources.
- *Manage patterns*. These use mashups as a vehicle for helping an organization leverage its existing assets more effectively.
- *Testing patterns*. These test mashup applications.

Both approaches are used in software design due to the advantages of the architectures that include them (Alur et al., 2003; Fowler, 2002; Erl, 2005; Gamma et al., 2005; Hanson, 2009), such as better integration and reusability, encapsulation, distribution, partitioning, scalability, performance, reliability, manageability, consistency and flexibility, packaging, and configurability. Other advantages are: support for multiple clients, independent development and faster development times.

Although traditional web applications and mashups are web applications that are built in the server and use the browser for displaying data, mashups have additional characteristics (Adams & Gerken, 2008): they aggregate disparate data, typically leverage public data, provide a common user interface for data, produce something new, produce something interesting, and add value by associating data.

Clearly, the problem of integrating a virtual campus and an external tool does not fall in the mashup category, because a virtual campus does not leverage public data and both the virtual campus and the external tool existed before they were integrated. However, their integration can take advantage of some user interface integration concepts belonging to the mashup pattern domain.

The architecture described in this paper implements mashup patterns in terms of multitier architecture patterns, taking advantages of both types of patterns to promote maintainability and other positive characteristics provided by pattern-based architectures. We have not found solutions of this type in the reviewed literature.

Finally, there are some software architectures for e-learning systems that we should mention: *OKI Open Service Interface Definitions* (OKI OSID) (OKI, 2011), *IMS Learning Information Services* (IMS LIS) (IMS, 2010a), *IMS Basic Learning Tools Interoperability* (IMS BLTI) (IMS 2010b) and *IMS General Web Services* (IMS GWS) (IMS, 2006). These standards are related to the integration of LMSs with other tools and are intended for the interaction between the virtual campus and the underlying LMS. However, in new generation virtual campuses, LMS core e-learning services tend to be wrapped by a virtual campus e-learning layer (Huertas & Navarro, 2013). Thus, the external tool usually interacts with this wrapping, which isolates LMSs from external tools. The use of these software architectures is not, therefore, of value for the type of integration described in this paper.

Integration architecture

This section describes the integration architecture proposed to provide a pattern-based solution to the problem of integrating host and guest applications. Although the detailed requirements have been listed in the introduction, the basic requirement is that the host application's user interface has to embed the guest application's user interface, providing a context for the uploading of content provided by the guest application. Therefore, the guest application has to be able to respond to the host via its integrated user interface.

In order to provide a homogeneous characterization of both applications, and taking into account the latest development architectures, let us suppose that both applications use a multitier architecture. Taking this restriction into account, Figure 1 depicts the architecture of the unrelated virtual campus and external tool using a UML 2 *component diagram* (Arlow & Neustadt, 2005).

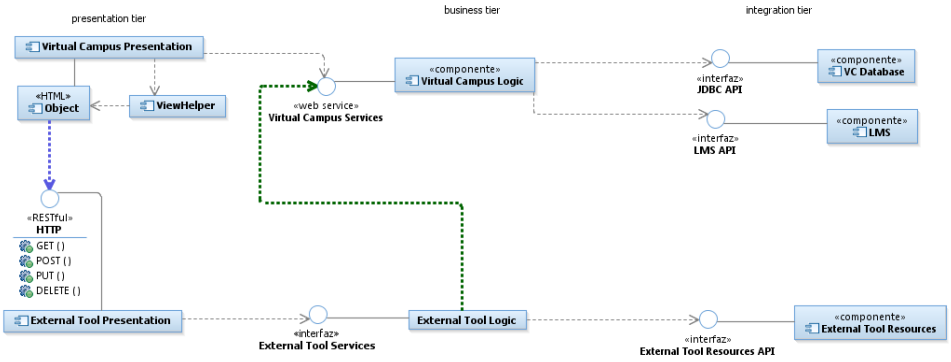

Figure 1. Proposed architecture to integrate virtual campus (host application) and external e-learning tool (guest application)

The integration layer of the Virtual Campus architecture is made up of an LMS and a virtual campus database exposed via their corresponding APIs. On this layer a set of management services and core e-learning services are built. The management services make it possible, for example, for a group of students enrolled in a class using the university online system to be automatically registered in an LMS course. The e-learning core functions are intended

to wrap the LMS-dependent functions into an LMS-independent system, thus isolating the virtual campus from the underlying LMS (Huertas & Navarro, 2013). If this component is not present in the architecture, it can be replaced by a *web service broker* (Alur et al., 2003) that mediates between the LMS and its clients. Finally, users of the virtual campus access their services through the user interface layer, built on the business service layer.

The architecture proposed for the external tool applies multitier architecture with integration, business logic and presentation layers, isolated via their corresponding interfaces.

In order to integrate the user interface of the guest application into the host application, the virtual campus uses the HTML element Object (W3C, 1999), which, assisted by a *view helper* (Alur et al., 2003), embeds the user interface of the guest application within its user interface. In this way, the *content aggregation* pattern of the mashup approach (Ogrinz, 2009) and the *presentation layer mashup* (Hanson, 2009) are implemented in the context of a multitier architecture (using *front* and *application controllers* (Alur et al., 2003)). Content aggregation is the mashup pattern most directly related to the integration of e-learning applications in the Ogrinz (2009) pattern catalogue.

Using this architecture, the presentation tier of the host application interacts with the guest application using the URLs that give access to its user interface directly, thus creating a RESTful-type interaction between the host application and the user interface of the guest application (note that in this figure, the interface with the REST stereotype only has HTTP methods available: GET, POST, PUT and DELETE (Hansen, 2007)). This is a very simple, functional solution. However, it has an important drawback: although it can be used from Apple Safari, Google Chrome, and Mozilla Firefox, it does not work with MS Internet Explorer. The user interface does not work properly in Microsoft's browser when a complex object is included in a web page, which makes the solution proposed in this paper unviable. However, we have not found this problem in the rest of the browsers previously mentioned.

For its part, the guest application does not need to interact with the host application using host user interface. The host *application services* (Alur et al., 2003) are therefore exposed directly, or using a *web service broker* (Alur et al., 2003), as web services to the external tool services.

This is, therefore, an *asymmetric* architecture: the host application interacts with the guest application using its user interface as RESTful web services (depicted in blue in Figure 1), while the guest application interacts directly with the host application services using SOAP or RESTful web services (depicted in green in Figure 1) (Hansen, 2007). Note that the asymmetry does not come from the nature of the web service implementation (SOAP vs. RESTful), but has to do with how the host application interacts with the guest application, via its user interface, while the guest application interacts with the host application via its business service layer.

This architecture is so generic that it could be applied to any two applications built using a multitier architecture that had to be integrated in line with the requirements listed in the introduction. Using this architecture, two main types of use case appear: those that are invoked by the user in the host application user interface to trigger an action in the guest application, and those that are invoked by the user in the guest application user interface (integrated in the context of the host application) to trigger an action in the host application.

In both types the same actions have to be performed:

- Information encoding.
- Application invocation and information sending.
- Information reception and decoding.
- Information processing.

However, there are differences due to the asymmetric architecture. In the case of invocation from host to guest the requests are made to the guest using its user interface. This forces a RESTful interaction between host and guest, as well as the adaptation of the code of the guest application to transmit the information belonging to the host application that has been received and has to be transferred to the business layer (e.g., a virtual campus user id).

In the case of invocation from guest to host, the requests are made to the guest using its business layer. Therefore, this interaction can be made using SOAP or RESTful web services, and it may be necessary to provide new services

in the business layer of the host application for dealing with the new request from the guest application (e.g., the uploading of a new type of content).

If both applications do not belong to the institution, the proposed approach could not be applied, because the initial requirements call for changes in both parts, at least in their user interfaces:

- The host application user interface has to embed the guest application user interface.
- Both applications need new widgets (e.g., buttons or links) for invoking the new functionality that appears as the result of the integration.

Example of integration

Integration of VCAA virtual campus and CREASE tool

This section provides a specific example of the integration architecture proposed: the integration of the VCAA virtual campus with the CREASE tool (the virtual campus with integrated tool is accessible at http://solaris.fdi.ucm.es/CVMB2/ with user demo and password demoKey). The main goal of the VCAA project was to develop a virtual campus independent of the underlying LMS responsible for implementing the core e-learning processes (Navarro et al, 2010). Thus, users interact with a generic virtual campus that hides the underlying LMSs. Figure 2 illustrates the VCAA architecture. According to this figure there are two main components: one responsible for core e-learning facilities and the other for management issues. These business processes are isolated from LMSs using the *VCAA canonical interfaces*, which are defined as SOAP web services. These interfaces are designed for the management of users, courses, resources, announcements, forums and platforms, and implemented for Blackboard Learn 9.1, Moodle 2.0 and Sakai 2.7.

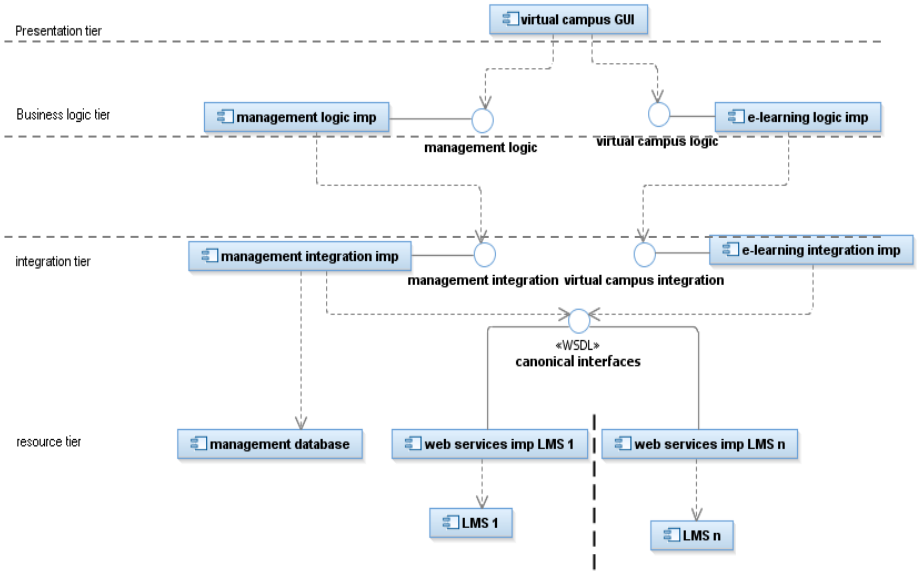

Figure 2. VCAA software architecture

This architecture has two main advantages: (i) platforms can be changed, and users are unaware of these changes; and (ii) whenever implementation for the VCAA Canonical Services is provided, platforms can be changed without the need to rewrite a single line of code of the virtual campus itself.

At present, this virtual campus has been deployed and is undergoing testing by three control groups of undergraduate and graduate courses, with a total of 91 users.

Our external tool is a search tool called CREASE (http://crease.lsi.uned.es), developed to cluster and organize unstructured didactic resources using *Formal Concept Analysis* (FCA), a mathematical theory that can be used for

the building of lattices of resources (Cole & Eklund, 1996; Ganter & Wille, 1999). The CREASE tool enables users to perform web searches for didactic resources, organizing them as lattices of resources depending on their descriptors. Thus, the developing framework of the CREASE search and classification tool is used to author learning content. To undertake this task, we first characterize instructional uses (i.e., goals) by means of a set of terms accurately defined by instructional designers. Secondly, we perform a number of search engine queries related to a domain keyword and enriched with the previous terms. Finally, the resources retrieved are analyzed and organized using FCA, in order to generate a concept lattice which could be explored by the user to select resources relevant to the initial pedagogical needs (Mayorga et al., 2012). As a working hypothesis, we assume that the resources retrieved by each query will be bound to a goal if there is sufficient overlap between the enriching terms in the queries and those characterizing the goal. Figure 3 shows how the components of the CREASE tool work together through the orchestrator component to perform these tasks.

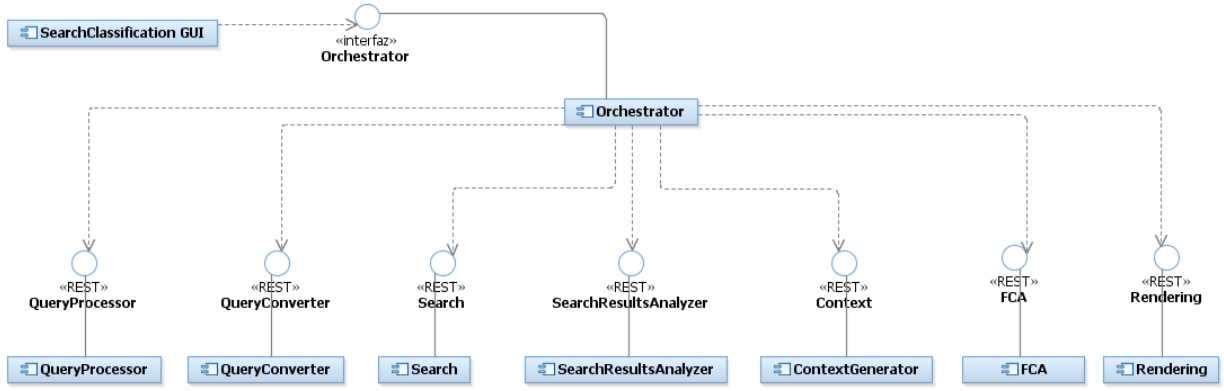

Figure 3. Search tool module dependences. Only presentation and business logic tiers are depicted

Architectures depicted in Figure 2 and 3 fit the architectures of the host and guest applications in Figure 1. According to this integration architecture, the goal is to obtain an integrated application such as the one depicted in Figure 4. In this figure the VCAA virtual campus user interface embeds the CREASE tool user interface. The CREASE tool searches for resources in the web, organizes them in lattices, and uploads these lattices in the VCAA virtual campus where the interaction originally started.

Thus, users from the virtual campus can launch the CREASE tool, search for resources in the web, and upload lattices of resources to specific virtual campus courses. Later, these lattices can be visualized in the context of the course using the CREASE tool. For example, Figure 4 illustrates how the CREASE tool shows the final concept lattice for the original query "Algebra" in the context of the VCAA virtual campus user interface. The diagram depicted is based on Hasse diagrams (Wille, 1982; Wille, 1992). The blue concept node represents the formal concept currently selected and the green ones represent those formal concepts which are closely related. The red formal concept is the top of the concept lattice and, finally, the green concept above the top formal concept represents a formal concept not comparable or not related to the formal concept that the user is currently exploring.

Using these two applications, this section provides a detailed design for the generic architecture of Figure 1, considering an implementation for the two main types of use which are of interest: those that are invoked by the user in the host application user interface to trigger an action in the guest application (use case uc1), and those that are invoked by the user in the guest application user interface (integrated in the context of the host application) to trigger an action in the host application (use case uc2).

Examples of these two main types are as follows:

- uc1: We consider the use case *show lattice*. This is a use case with its origin in the virtual campus user interface. Thus, a lattice previously uploaded to a course is displayed using the integrated CREASE tool user interface, as shown in Figure 4.
- uc2: We consider the use case *upload lattice*. This is a use case with its origin in the integrated CREASE tool. A search lattice built by the CREASE tool is thus uploaded to the virtual campus course, where the interaction with the CREASE tool originally started.

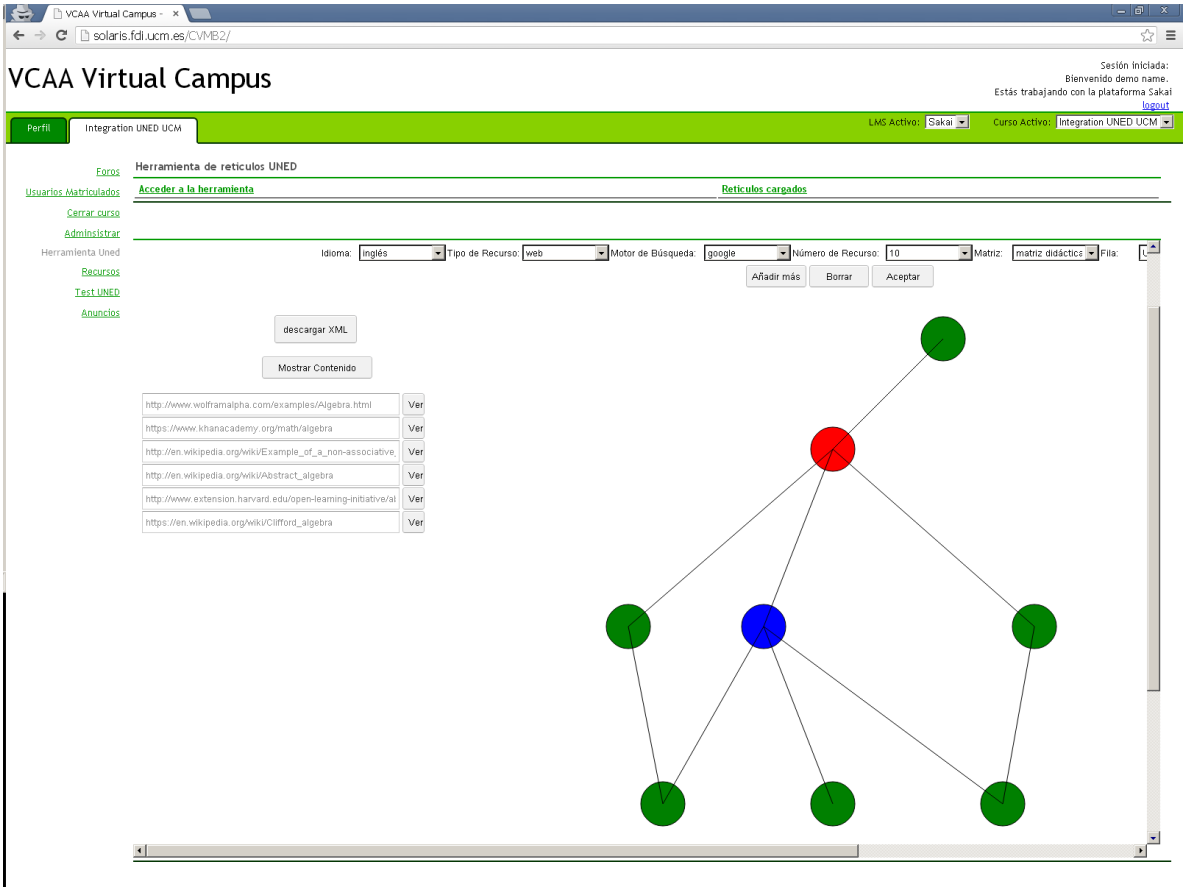

Figure 4. CREASE tool (pointed blue) integrated within the VCAA Virtual Campus (pointed green)

Uc1 show lattice

Figure 5 depicts the detailed class diagram for the presentation tier of the host application sketched in Figure 1 using a *UML 2 class diagram* (Arlow & Neustadt, 2005) enhanced with *UML WAE* notation (Conallen, 1999). Basically, according to UML WAE notation, client pages represent contents directly retrieved by the web server (e.g., HTML pages), server pages represent computational resources run by an extension to the web server (e.g., Java servlets run by Tomcat), client pages can be built by server pages and navigated associations represent hyperlinks.

The key issue is to be able to include the CREASE tool in the context of the Virtual Campus user interface. In order to solve this issue, the HTML element Object is used. This element allows an HTML page to be loaded inside another HTML page, enabling a nested interaction. The virtual campus user interface uses a view helper to build its user interface, including an HTML Object element with the URL of the CREASE tool. This URL includes encoded information about the virtual campus session (course id, user id, etc.) as well as information about the lattice.

The interaction between object instances of these classes for implementing use case uc1 is depicted in Figure 6 using *a UML 2 sequence diagram* (Arlow & Neustadt, 2005). In this figure the view helper builds the URL that invokes the CREASE tool user interface in the context of an Object element nested in the VCAA virtual campus user interface. The information is encoded using a dedicated class and the Object element with the target URL is printed in the HTML page generated.

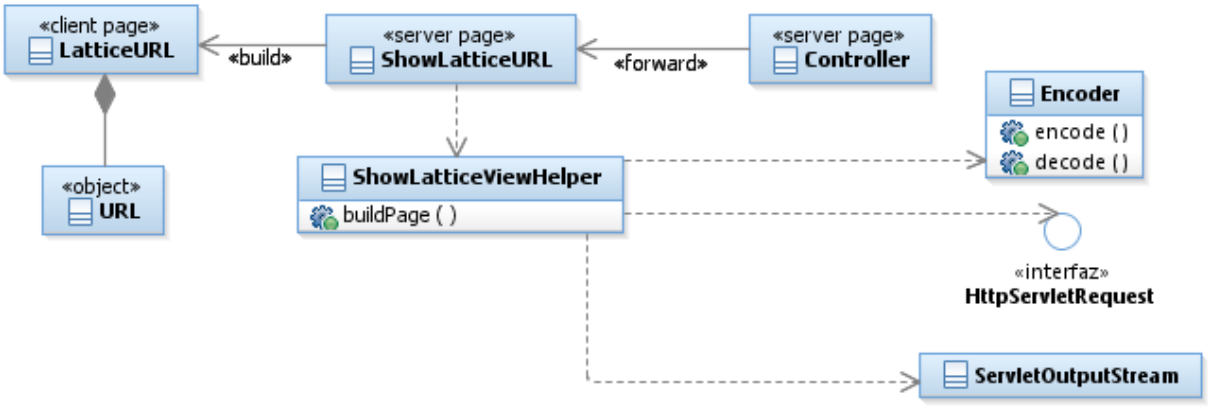

Figure 5. Detailed design of the presentation tier for the use case uc1: show lattice

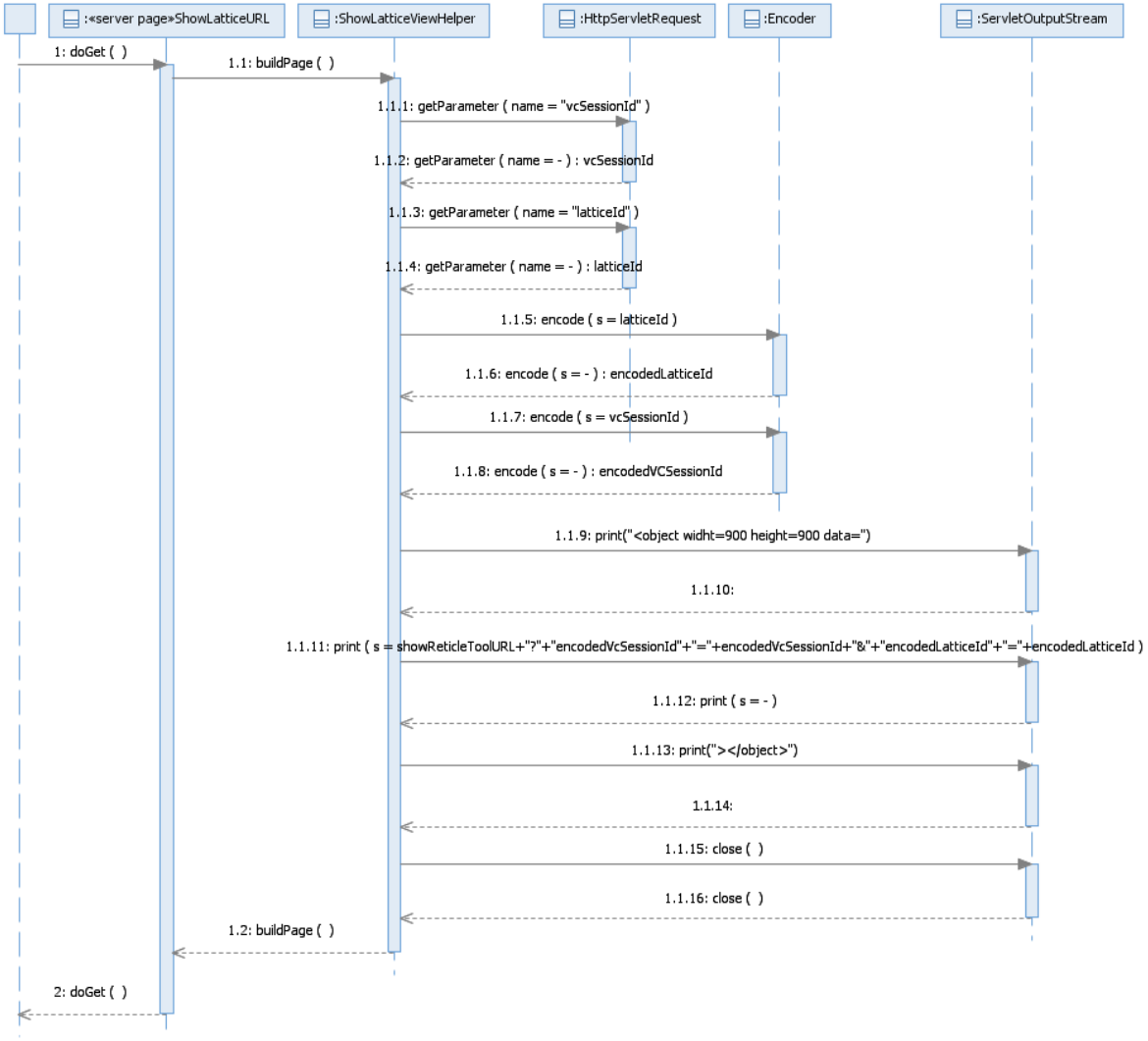

Figure 6. Virtual campus user interface including the nested Object element that loads the CREASE tool responsible for showing the lattice in the VCAA virtual campus user interface

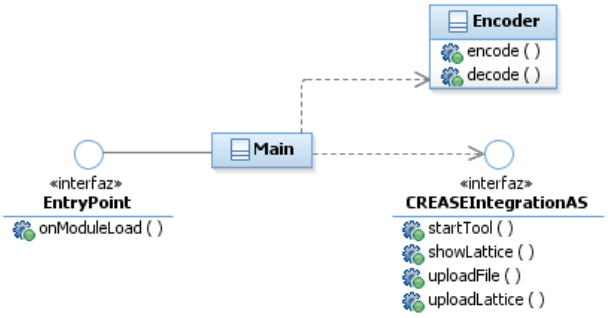

Figure 7. Main is a GWT entry point for the implementation of use case uc1

For the management of this invocation, submitted via an HTTP GET request, the CREASE tool user interface has to decode the information received, and to display the lattice as it would be displayed by the application running as a standalone application. The CREASE tool is a J2EE application whose user interface has been built using Google GWT (Tacy et al., 2013). Therefore it is necessary to define a GWT EntryPoint for the invocation of the component responsible for the implementation of use case uc1. Figure 7 depicts this entry point (called Main.)

As Figure 8 shows, when this entry point is invoked from the VCAA Virtual Campus, it receives the information sent (virtual campus session id plus the lattice), decodes it, stores the virtual campus session id in the HTTP session, shows the lattice, delegating to the application service defined in Figure 7.

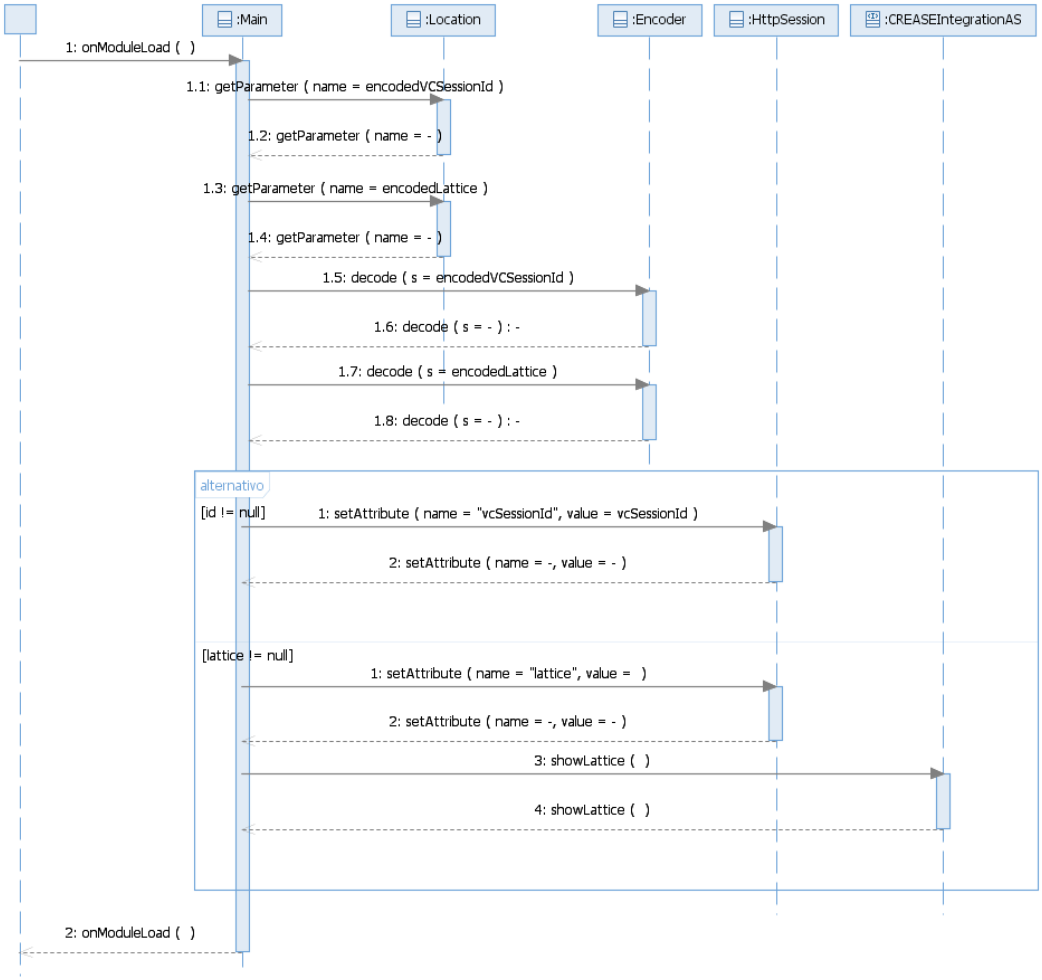

Figure 8. Interaction at the GWT entry point for the implementation of use case uc1 in the CREASE tool

Uc2 upload lattice

For the implementation of use case uc2 (upload lattice) the CREASE tool simply needs to invoke the web services provided by the VCAA Virtual Campus using the WSDL web services provided by VCAA Virtual Campus. Figure 9 depicts the CREASE application service responsible for implementing this use case.

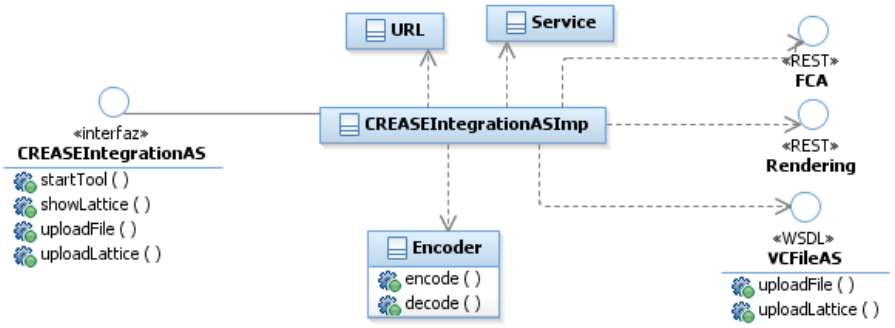

*Figure 9***.** Detailed design of the business logic for the implementation of use case uc2

When the application needs to upload a file or a lattice to the VCAA Virtual Campus, the CREASEIntegrationASImp class extracts the necessary information from the session (virtual campus id as well as the lattice of the file that has to be uploaded), encodes the information, and invokes the implementation of the WSDL VCAA web services using the JAX-WS proxies (Hansen, 2007). Figure 10 shows this interaction.

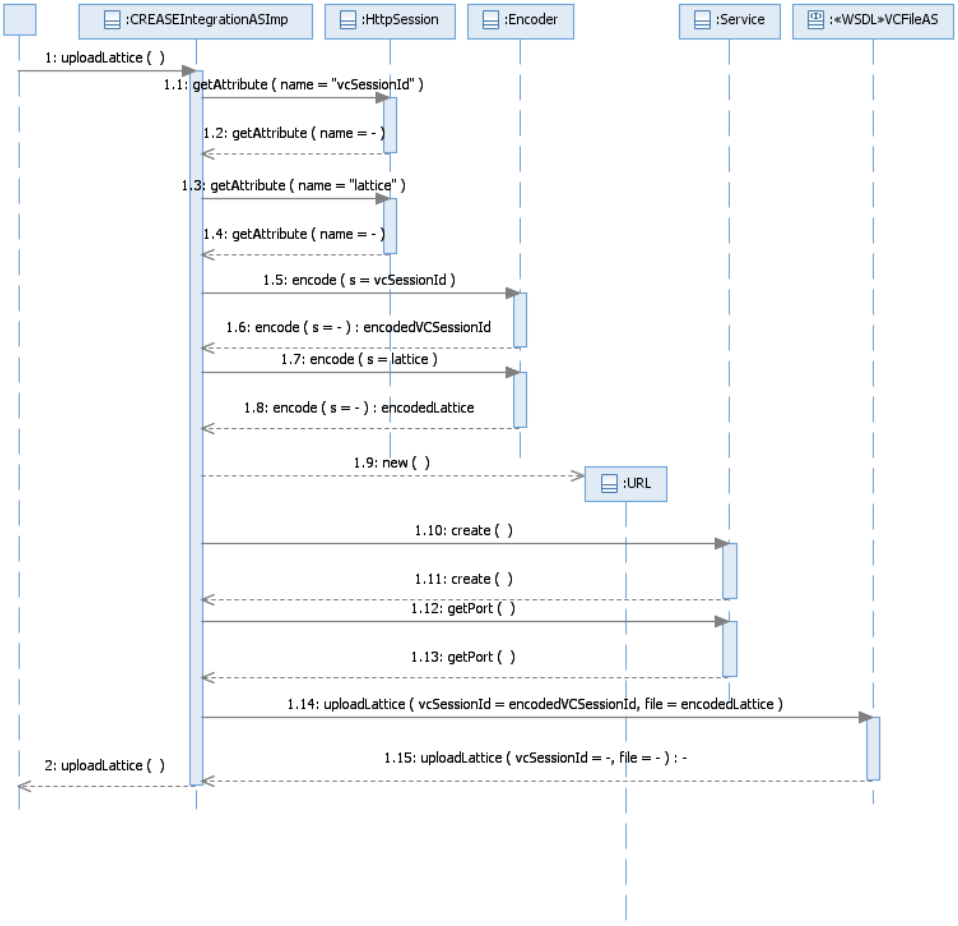

Figure 10. Interaction in the CREASE tool for lattice uploading in the VCAA Virtual Campus

Discussion

This paper focuses on the software component of application integration. However learners and educators are the main beneficiaries of the solution. For example, in the context of a large virtual campus (such as the UCM virtual campus) particular users from specific Faculties or departments may need specialized learning object repositories to handle the unique metadata from each domain (Navarro et al., 2013). It would be desirable to have a generic integration architecture that is able to combine those learning object repositories and virtual campuses. Learning objects could thus be built using the specific tools from each domain and these objects could then be integrated in the virtual campus.

Therefore, educators should be aware that they would be able to develop tools for their specific learning needs and that these tools could subsequently be integrated in virtual campuses, making it easier to provide domain-specific learning objects in generic virtual campuses.

Examples of these tools are almost endless, provided that they can be integrated in the architecture described in this paper: repositories of learning objects, tools that promote collaborative and constructive learning or simulators. Tools that can be integrated in virtual campuses can be conceived as *satellite tools* (Fernández-Valmayor et al., 2011).
From an educator's point of view this paper provides a software architecture that can integrate software applications into a single user interface.

In simple terms we could divide the software application into two parts: the part responsible for interacting with the user through a user interface (the *view*) and the part responsible for processing the input provided by the user (the *model*, which in this paper has been split into business logic –processes- and the integration layer -data access-, in line with multi-tier architectures). In traditional integration architectures a software application (let's call it $application_1$, with $view_1$ and $model_1$), interacts with another software application ($application_2$, with $view_2$ and $model_2$) using both models in a coordinated manner to obtain $application_{1'}$, with view $view_{1'}$ and model $model_1+model_2$.

With the approach presented in this paper a new software application is created that not only integrates models but also views, which could be expressed as $aplication_{12}$, with view $view_1+view_2$ and model $model_1+model_2$.

This integration has an important advantage: the complex user interface (view) needed to access an application's business logic (which is part of the model) can now be reused in a coordinated manner with the user interface of another application. Traditional integration architectures mainly focus on model integration and not on views integration.

There is, however, one important drawback to this solution: programmers must have access to both applications in order to integrate them because the user interface of the first application, the host application, has to include the user interface of the second application. Conversely, the guest application has to include facilities for interacting with the host application. This cannot be resolved without access to the source code of both applications.

Conclusions and future work

Nowadays virtual campuses have become complex software applications that enlarge LMS functions with services needed by educational institutions or by users.

In these campuses, users demand the enhancement of the learning possibilities provided by classic LMSs with the use of external e-learning tools, such as those designed for the definition, search and management of complex learning objects. However, in most cases, virtual campuses and external e-learning tools are developed independently.

This paper describes a pattern-based integration architecture for virtual campuses with external e-learning tools. According to this architecture, virtual campus users directly interact with the tool thanks to the inclusion of the tool in the virtual campus user interface using the HTML Object element, and some REST-inspired techniques for communication between both applications. The external tool interacts with the virtual campus using a set of services published as web services (SOAP or RESTful).

This solution is therefore based on an asymmetric architecture: the virtual campus exposes its services as web services, whereas the external tool is directly integrated in the context of the user interface of the virtual campus. This asymmetry is necessary because there is no sense in exposing the external tool services as web services and then rebuilding the external user interface in the context of the virtual campus, as the direct inclusion of the user interface of the external tool is a simpler and more straightforward solution. In addition, the presence of the virtual campus services as web services facilitates the integration of the virtual campus in the context of a large educational institution, where integration needs are very significant. The proposed architecture is contextualized in terms of e-learning applications, considering virtual campuses as host applications and external tools as guest applications, but it could be reused in other applications with similar requirements.

Regarding the integrated interaction between host and guest applications, there are differences due to the asymmetric architecture. In the case of invocation from host to guest the requests are made to the guest using its user interface. This forces a RESTful interaction between host and guest, as well as the adaptation of the code of the guest application to transmit the information belonging to the host application that has now been received and has to be transferred to the business layer (e.g., a virtual campus user id).

In the case of invocation from guest to host, the requests are made to the guest using its business layer. Therefore, this interaction can be made using SOAP or RESTful web services, and it may be necessary to provide new services in the business layer of the host application to deal with the new request of the guest application (e.g., the uploading of a new type of content).

The architecture is based on the integration of mashup and multitier architecture. The presence of these patterns guarantees design maintainability as well as other positive properties. In the proposed solution the mashup pattern's content aggregation and presentation layer mashup are implemented using a view helper multitier pattern that builds the URL used by the embedded HTML Object of the host application user interface. Other multitier patterns used in this integration architecture are front and application controller, application service and web service broker. Obviously the rest of the twenty one multitier patterns (Alur et al., 2003) can still be used in the architecture.

This architecture has been validated by integrating an FCA-based search and classification tool in a virtual campus. Thus, users of the virtual campus can use this tool in the context of a virtual campus course, reusing the content extracted by the tool.

Currently, the VCAA Virtual Campus is being tested by two university courses with 91 users, with promising results. Future work includes the development of more functionality in the VCAA Virtual Campus, and its integration with other external e-learning tools. In addition, we are working on a solution to enable the integrated use of both tools in Microsoft Internet Explorer and validation of the approach with a broader range and number of users.

Acknowledgements

Ministerio de Ciencia e Innovación (projects AACV TIN2009-14317-C03-01 and CREASE TIN-2009-14317-C03-03), Universidad Complutense de Madrid (group 921340), and e-Madrid project (S2009/TIC-1650, Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid) have supported this work.

References

Adams, H., & Gerken, J. (2005). *Choosing between mashups and traditional web applications*. Retrieved December 13, 2013, http://www.ibm.com/developerworks/lotus/library/mashups-web/

Allison, D. H., & DeBlois, P. B. (2008). The EDUCAUSE current issues committee. *EDUCAUSE Quarterly, 31*(2), http://www.educause.edu/ir/library/pdf/eqm0823.pdf

Alur, D., Crupi, J., & Malks, D. (2003). *Core j2ee pattern: Best practices and design strategies* (2nd ed.). Palo Alto, CA: Prentice Hall.

Arlow, J., & Neustadt, I. (2005). *UML 2 and the unified process: Practical object oriented analysis and design* (2nd ed.). Upper Saddle River, NJ: Addison-Wesley Professional.

The Campus Computing Project Survey (2012). Retrieved December 13, 2013, http://www.campuscomputing.net/sites/www.campuscomputing.net/files/Green-CampusComputing2012.pdf

Cole, R. J., & Eklund, P. W. (1996). Applications of formal concept analysis to information retrieval using a hierarchically structured thesaurus. In P. Eklund et al. (Eds.), *Conceptual structures: Knowledge Representation as Interlingua* (pp. 1-12). New York , NY: Springer.

Epper, R. M., & Garn, M. (2004). The virtual university in America: Lessons from research and experience. *EDUCAUSE Centre for Applied Research (ECAR) Research Bulletin, 2004*(2). Retrieved from http://www.educause.edu/ir/library/pdf/ERB0402.pdf

Erl, T. (2005). *Service-oriented architecture (SOA): Concepts, technology, and design*. Upper Saddle River, NJ: Prentice Hall.

Etzion, O., & Niblett, P. (2010). *Event processing in action*. Greenwich, CT: Manning Publications.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Boston, MA: Addison-Wesley Professional.

Fernández-Valmayor, A., Fernández-Pampillón Cesteros, A. M., Fernández-Chamizo, C., Navarro, A., & Cristobal, J. (2011). Implantación de un campus virtual de grandes dimensiones: el campus virtual de la UCM [Deployment of a Large Scale Virtual Campus]. *IEEE-RITA, 6*(4), 167-174.

Ganter, B., & Wille, R. (1999). *Formal concept analysis – mathematical foundations*. Berlin, Germany: Springer Verlag.

Green, K. C. (2005) *The 2005 national survey of IT in U.S. high education: Growing campus concern about IT security; slow progress on IT disaster planning*. Retrieved from http://www.campuscomputing.net/sites/www.campuscomputing.net/files/2005-CCP.pdf

Hanson, J. J. (2009). *Mashups: Strategies for the modern enterprise*. Addison-Wesley Professional.

Hansen, M. D. (2007). *SOA using Java web services.* Upper Saddle River, NJ: Prentice-Hall.

Huertas, F., & Navarro, A. (2013). Integration mechanisms in e-learning platforms. *International Journal of Computer Information Systems and Industrial Management Applications, 5*, 714-721.

IMS (2006), *IMS general web services specification - version 1*. Retrieved December 13, 2013, from http://www.imsglobal.org/gws/index.html

IMS (2006), *IMS learning information systems specification - version 2.0*. Retrieved December 13, 2013, from http://www.imsglobal.org/lis/index.html

IMS (2006), *IMS basic learning tools interoperability specification - version 1.0*. Retrieved December 13, 2013, from http://www.imsglobal.org/lti/index.html

Juric, M. B., Basha, S. J., Leander, R., & Nagappan, R. (2001). *Professional J2EE EAI*. Hoboken, NJ: Wrox Press.

Mayorga, J., Vélez, J., Cigarrán, J., & Rodriguez-Artacho, M. (2012). An architecture for retrieving and organizing web resources for didactic purposes. *Journal of Research and Practice in Information Technology, 44*(2), 183-201.

Navarro, A., Cristóbal, J., Fernández-Valmayor, A., Fernández, C., Hernanz, C., Guillomía, S., & Buendía, F. (2010). Towards a new generation of virtual campuses. In T. Atmaca, J. Palicot, A. Nafkha, T. Tsiatsos, M. Marot, & O. Dini, *Sixth Advanced International Conference on Telecommunications* (pp. 70-73). Los Alamitos, CA: IEEE Computer Society.

Navarro, A., Cristóbal, J., Fernández-Chamizo, C., & Fernandez-Valmayor, A. (2012). Architecture of a multiplatform virtual campus. *Software Practice and Experience, 42*(10), 1229-1246.

Navarro, A., Rodríguez-Artacho, M., Huertas, F., Cigarrán, J., & Buendía, F. SOA integration of a tool for retrieving open learning resources into a modular virtual campus. In M. B. Nunes & M. McPherson (Eds.), *IADIS e-learning 2012* (pp. 262-269). Lisbon, Portugal: IADIS.

Navarro, A., Fernández-Pampillón Cesteros, A. M., Fernández-Chamizo, C., & Fernández-Valmayor, A. (2013). A meta-relational approach for the definition and management of hybrid learning objects. *Educational Technology & Society, 16*(4), 258-274.

Ogrinz, M. (2009). *Mashup patterns: Design and examples for the modern enterprise*. Boston, MA: Addison-Wesley Professional.

OKI (2011). *Open services description interfaces repository*. Retrieved from http://osid.org/specifications/osid/package.html

PLS RAMBOLL (2004). *Studies in the context of the e-learning initiative: Virtual models of European universities.* Retrieved from http://firgoa.usc.es/drupal/files/virtual_models.pdf

Tacy, A., Hanson, R., Essington, J., & Tokke, A. (2013). *GWT in action* (2nd ed.) Greenwich, CT: Manning Publications.

Van Dusen, G. C. (1997). The virtual campus: Technology and reform in higher education. *ASHE-ERIC Higher Education Report, 25*(5). Retrieved from http://files.eric.ed.gov/fulltext/ED412816.pdf

Weaver, J. L., Mukhar, K., & Crume, J. P. (2004). *Beginning J2EE 1.4: From novice to professional*. New York, NY: Apress.

Wille, R. (1982). Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, *Ordered Sets* (pp. 445-470). Dordrecht, Holland: D. Reidel Publishing Company.

Wille, R. (1992). Concept lattices and conceptual knowledge systems. *Computers and Mathematics with Applications*, *23*(6), 493-515.

World Wide Web Consortium. (1999). *HTML 4.01 specification*, 1999. Retrieved December 13, 2013 from http://www.w3.org/TR/1999/REC-html401-19991224/

Yábar, J. M., Hernández, J., López Roldán, P., & Castellá, J. (2007). The UAB virtual campus: An essential platform for a European higher education environment. *Journal of Cases on Information Technology*, *9*(2), 37-48.