Adding Process-Driven Collaboration Support in Moodle

Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon
University of Vigo, rperez@gist.det.uvigo.es, mcaeiro@det.uvigo.es, lanido@det.uvigo.es

***Abstract* - Moodle is a well-known open-source LMS. Moodle approach to Collaborative Learning is just limited to "putting people around a table". It provides a means for participants to interact with other participants (other people around the same table) and with Learning Contents (the objects on the table). In this scenery, by no means it can be guaranteed that collaboration will occur. In order to assure that real collaboration will take place, we define workflow-like collaboration structures which we name *learnflows*. In this paper, we present a conceptual architecture and a concrete implementation for the support of *learnflows* in Moodle. We design a *Learnflow Engine* on top of a generic-purpose *Workflow Engine*, just by using out-of-the-box workflow functionality. This *Learnflow Engine* is exposed as a Web Service, enabling the consumption of *Learnflow Web Service Methods* from the Moodle integration code, which is designed following *Aspect-oriented Software Development* techniques.**

Index Terms – Moodle, Collaborative Learning, jBPM

INTRODUCTION

For the last few years, Learning Technology has become a strategic sector both for enterprises and higher education institutions. Learning Technology is a catch-all term that encompasses numerous concepts such as distance learning, computer-supported learning, and many others. Despite of recent advances in Learning Technology standardization, e-learning systems, both proprietary and open-source ones, are being developed with not enough interoperability support. The consequences of this approach are that each new system is developed from scratch, the systems are not compatible, the Learning Contents are not portable between systems, and the worst of all: Learning Technology is permanently stuck at the same stage of development. In this way, some interesting paradigms, such as Collaborative Learning, have no adequate support in current Learning Technology systems.

The ideas in Learning Technology have been usually materialized into Learning Management Systems (LMSs). LMSs put into contact participants with Learning Contents and with other participants. Moodle [1] has become the de-facto standard in web-based LMSs [2]. Thousands of higher education institutions are currently using it to satisfy their learning needs. Moodle is easy to install and manage. Another virtue of this system is its GPL license, which allows everyone to use it free.

Regarding the support of Collaborative Learning, Moodle counts with a number of collaborative tools. In this group there are tools such as forums and chats, which allow participants to interact among them. This kind of tools enables free collaboration among participants. Notwithstanding, as found in the literature, free collaboration does not systematically produce learning [3]. In order to enhance the effectiveness of Collaborative Learning, interactions among participants, as well as the information they interchange, have to be structured [4]. The term learnflow is used in the learning domain (making an analogy with the term workflow [5]) to describe such well-ordered collaboration structures. Both learnflows and workflows coordinate users at an activity level, defining a process as a sequence of activities.

There are different alternatives for supporting learnflows. Since we formalize collaboration structures as processes (learnflows), these processes can be computationally represented using a generic-purpose process definition language and they can be executed in a generic-purpose Workflow Engine [6]. This approach is optimal regarding interoperability and reusing purposes.

In this paper, we describe a proposal for the integration of jBPM Workflow Engine [7] in Moodle following state-of-the-art techniques in application integration. Figure 1 shows a simple view of this architecture. Firstly, it is explained how the Workflow Engine is exposed as a Web Service, making the workflow API available to be invoked from the Moodle side. Secondly, it is explained how the integration is done at the Moodle side following *Aspect-oriented Software Development* (AOSD) techniques [8] (new crosscutting concerns are added in Moodle) to identify the points at which the workflow services have to be consumed.

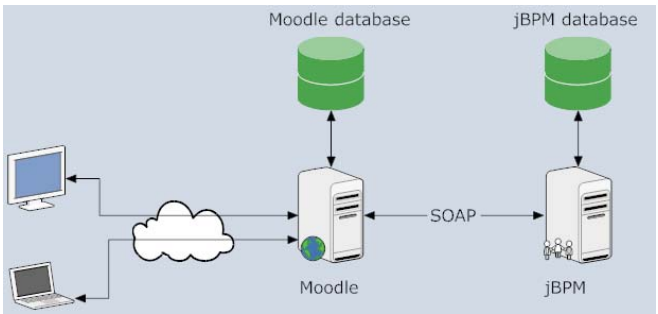

Figure 1. A simple view of the architecture

MOODLE

In our study of current Learning Technology tools, we have noticed that open-source ones are gaining a wide acceptance in higher education institutions. Moodle is by far the most used open-source Learning Management System, with millions of users (moodlers) worldwide. Its GPL license, stability, and the great community built around it, are the main reasons for its popularity.

When Martin Dougiamas designed Moodle, he chose a programming language that could be easily learnt by people, with the purpose of creating a great community of developers. The chosen language, PHP, is mostly used to program Web sites. For persistence, Moodle uses a MySQL database, which is open source.

We evaluate Moodle following the principle of separation of concerns as follows:

- From a structural point of view the way in which Moodle structures content is very rigid, since it only allows to design courses composed of sections into which the Learning Contents are aggregated. The content structure is a hierarchy: course, section, and resource.
- From an organizational point of view, Moodle allows to assign predefined roles to users. The roles which Moodle allows are: administrator, teacher, and learner. In Moodle 1.9 it is allowed to define roles in tools. In this way, a certain user can be an administrator in a given forum and a learner in the entire course. The differences between one role and another are the permissions assigned to each one.
- From a temporal point of view, Moodle allows to liberate contents by weeks: each week a section is liberated. Moodle does not allow to define concrete points in time to liberate contents.
- From an ordering point of view, Moodle does not allow to sequence sections or resources.

PROBLEM DESCRIPTION

Several functionalities which are lacking in Moodle are necessary to support learnflow definition and execution [9] [10]:

- Capability to assign different activities to different participants in the same learnflow
- Capability to mark the end of an activity
- Capability to sequence activities

We describe learnflows with an example (see Figure 2). In this example it can be seen how the four activities which compose the learnflow are sequenced and assigned to four different roles. Activity 1 is the first to be performed, followed by activities 2 and 3, which are performed in parallel. Finally, activity 4 is performed.

The activities in the example correspond to the sections at the Moodle course page. In this way, section 1 is performed by swimlane 1 in first place. In second place, activities 2 and 3 are performed in parallel by roles 2 and 3. Finally, section 4 is performed by role 4. With this approach, sections are performed collaboratively by four different participants.

Therefore, our problem is to extend Moodle in order to support both the definition and execution of learnflows. We evaluate the implications for Moodle in the two following subsections, in relation with the data model and the functionality.

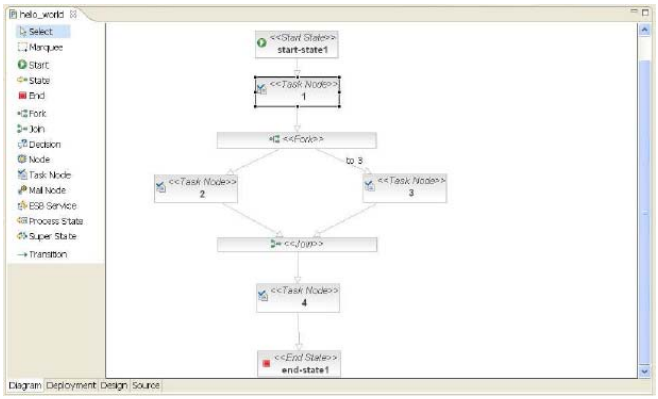

Figure 2. An example of learnflow

ARCHITECTURE

We determined to follow a Service Oriented approach to deal with the integration of the two information systems (Moodle and the Workflow Engine). Hence, the functional control can be passed from Moodle to the Workflow Engine by means of a Web Service invocation, and after the Workflow Engine has completed its processing, the functional control returns to Moodle. Web Services overcome the dependency on programming languages and execution platforms by being built over accepted standards such as XML, SOAP, WSDL, and UDDI. Therefore, the architectural design can be considered as a platform-independent model, which may have several platform-dependent implementations, depending on the concrete technologies of choice, among them, the chosen Workflow Engine. Hereafter, we suppose that the Workflow Engine is written in Java, since most of the Workflow Engines we have analyzed are integrated into J2EE infrastructure. Our approach does not lose generality, since there are libraries for publishing Web Services available in most of programming languages.

The overall architecture is Service Oriented (Service Oriented Architecture, SOA), since the workflow functionalities are published as Web Services. This architectural pattern has several advantages:

- Moodle has a module to facilitate Web Service consumption, namely NuSoap [11]
- There exist tools to develop Web Services in almost every programming language which facilitate to

publish/consume Web Services, acting as an abstraction layer from SOAP and low-level protocols.

- The Workflow Server can reside in a remote place over the Internet, since SOAP works over HTTP, overcoming firewalls and port/protocol restrictions.

Figure 3 shows a diagram of the architecture as a layered system:

- At the left side of the figure, we can see how an *Aspect Layer* is placed on top of Moodle, containing the integration logic. The *Middleware Layer* is composed by a SOAP engine, specifically NuSOAP.
- At the right side, we can see the *Workflow Layer*, which is composed by jBPM. Over the *Workflow Layer*, and using the *Workflow Service Methods* provided by jBPM, a *Learnflow Layer* has been built. On top of the right side, it is placed the *Middleware Layer*, which is composed of Axis SOAP Engine.

We detail the architecture starting with the *Learnflow Web Service*. The three layers into which we decompose the *Learnflow Web Service* are detailed in this order: the *Workflow Layer*, the *Learnflow Layer*, and the *Middleware Layer*. Further, we detail the architecture of the Moodle side of the system.

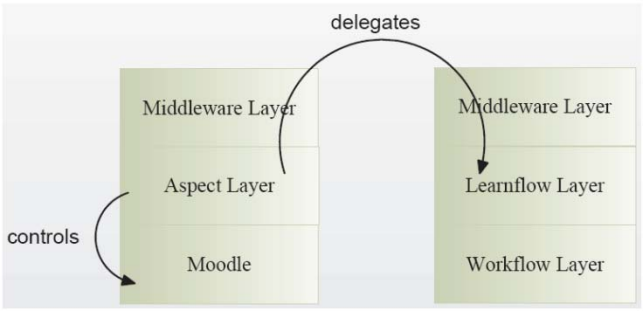

Figure 3. A view as a layered system

I. *The workflow layer*

Most of the Open Source Workflow Engines provide very similar functionalities. Our election of a concrete Workflow Engine is driven by the stable/production projects which are using that Workflow Engine. Under these criteria, jBPM is the most accepted Workflow Engine.

In this subsection it is detailed how the *Workflow Service* Methods are accessed from the *Learnflow Layer*. jBPM uses a *Factory Method Patter* for creating the service-provider object, the *JbpmContext*. This *creational pattern* is useful when, like in this case, the object creation process depends on settings in configuration files.

The *JbpmConfiguration* is a thread safe object and serves as an object factory for *JbpmContext*. All threads can use the *JbpmConfiguration* as a factory for *JbpmContext* objects. Thus, a *JbpmConfiguration* can be used to create *JbpmContext* (one per user request).

The *JbpmContext* class provides access to the most common operations. A *JbpmContext* represents one transaction. *Workflow Service Methods* are available inside a context block.

II. The learnflow layer

The *LearnflowEngine* implements the *Learnflow Service Methods*, making use of the *Workflow Service Methods* provided by the *Workflow Layer*. The *Learnflow Engine* customizes jBPM for the participants in the Learning Technology system. The *LearnflowEngine* class implements the *Learnflow Service Methods* making use of jBPM API, which is invoked by means of the *JbpmContext* class. This design offers two main advantages:

- It provides the *Learnflow Engine* with a specific interface for the learning domain. Since SOAP is based on message-passing, the parameters of the LearnflowEngine methods must be as simple as possible, since it is not allowed to pass Java objects into SOAP messages.
- It enables the binding of Web Service methods to *LearnflowEngine* methods in a straightforward way.

III. The middleware layer

In order to facilitate the consumption of *Learnflow Service Methods* from the Moodle side we make use of the functionalities provided by Apache Axis [12]. Apache Axis is a *SOAP engine*, a framework for constructing SOAP servers, clients, etc. Apache Axis generates a WSDL file and a skeleton class from the interface definition in Java.

The *JavaToWSDL* tool provides for automatic WSDL generation from Java code directly. Thus, *JavaToWSDL* facilitates to declare a Web Service interface as a Java class, freeing the developer from low-level details. The WSDL file is automatically generated from the Java class containing the declaration of Web Service methods as a Java interface.

III. The aspect layer

In our approach, we place two layers over Moodle. Directly on top of Moodle it is placed the *Aspect Layer*, which is composed by *aspects*, that is, snippets of code which are run from scattered places throughout the Moodle source code, each one of them corresponding to a specific *concern,* such as the ones shown in Figure 4. On top of the *Aspect Layer* we place a *Middleware Layer* which facilitates the consumption of *Learnflow Web Service Methods*.

The *Aspect Layer* defines an *aspect* for each *Learnflow Web Service Method*. For example, the *getPendingTasks()*, *endOfActivitySignaling()*, and *isUserAllowed() Learnflow Web Service Methods* are each one associated with an *aspect*.

The concrete AOP technique we are using is known as *code advising*, and it refers to the ability to intercept

application events. *Code advising* is used to control the *Moodle layer*. In this way, certain operations can be blocked or released depending on the *Learnflow Web Service* control.

In our approach, the *Aspect Layer* does not implement the learnflow-related logic, but it delegates this function to the *Learnflow Web Service*. This practice allows to decouple the integration code composed by the *Aspect Layer* from the concrete implementation of the *Learnflow Engine*.

The advice piece of code is composed by a Web Service invocation. We use the *NuSOAP* library, which facilitates the consumption of Web Service methods.

Following the AOSD approach, it remains to specify *when* the *advice* part of code has to be executed. In this case study (signaling the end of an activity), the *advice* has to be executed *after* the participant completes a Moodle activity.

The *getPendingTasks() Learnflow Web Service Method* is invoked from another advice. To fully specify this concern, we have to define *when* it has to be executed. In this case, the *advice* has to be executed *before* the participant gets the course main page.

The *isUserAllowed() Learnflow Web Service Method* has to be invoked *around* the activity page: if the participant is not allowed to perform the activity, the activity must not be displayed. In this case, the activity part of code is bypassed, since the participant is not allowed to see that information.

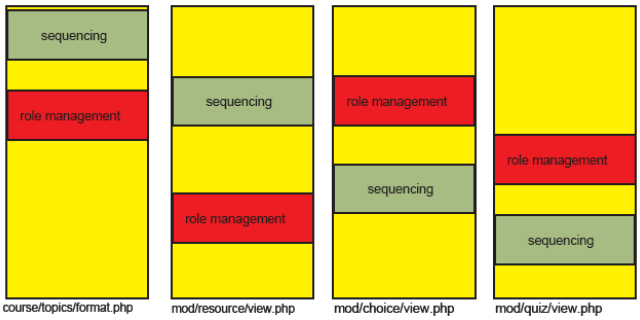

Figure 4. Concerns which crosscut several Moodle modularization units

LESSONS LEARNT

In Learning Technology, collaboration support can be enhanced by adopting a process-based approach. In this way, the interaction of learners with teachers, with other learners, and with learning activities, is formalized as a learning process, or as named in this paper: a *learnflow*.

In our approach, a *domain-specific* process (a *learnflow*) is translated into a *domain-general* process without loss of information. We use jPDL as a *process lingua franca* for *domain-general* computational representation. Thus, we can run the *learnflow* using a generic-purpose Workflow Engine. The customization of the generic-purpose Workflow Engine is done by adopting a *facade pattern* over the Workflow Engine. In this way, the functionalities provided by the *Learnflow Engine* are achieved just by using out-of-the-box jBPM functionalities. This approach results into a *Learnflow Engine*, whose capabilities depend on the underlying Workflow Engine (in this case, the capabilities of jBPM Workflow Engine). For example: at times it is desirable to modify slightly a learnflow at run-time without having to undeploy and redeploy. Regrettably, jBPM does not provide those flexibility features, as well as the other assessed Workflow Engines.

In the last paragraph it is said that our approach is suitable for supporting lots of domain-specific processes which are currently being developed with programming languages such as Java. Moreover, with our approach processes are *first-class objects*: processes can be created, instantiated, deployed, accessed to, etc. A process is a *first-class object* like objects in a OOP language. This approach is richer than to code the process logic with a programming language.

When dealing with systems integration, it is required to provide the components with a common and standardized interface, at times known as the *Enterprise Service Bus*. Web Services are particularly suitable for providing legacy components with a standardized interface. In this way, Web Services are the natural replacement for technologies such as CORBA and JavaRMI. A Web Service can act as a wrapper over a legacy component or, as the case presented at this paper, it can provide an application with a new kind of interface.

At times, it can be simply not possible to do some processing in the same environment used to run a web site (typically, a Web Server + PHP interpreter). In that case, it may be worth to delegate that processing into a remote system. Web Services are a very valuable technology for that. In this way, the execution flow of a legacy application can be broken down and transferred to a remote system by means of a Web Service invocation. The main drawback in this approach is that Web Services do not provide us with a systematic methodology for designing a complex intervention in legacy code, maybe consisting in lots of Web Service invocations throughout the legacy source code. In that case, we can implement an a*dvice* as a Web Service invocation, delegating the advice-related processing to the remote system. This procedure is named as *aspectual services*.

This work has also highlighted the lack of *separation-of-concerns*} in Moodle. PHP in itself is not a Model View Controller (MVC) framework. In consequence, web sites based on PHP usually present a poor separation of concerns. This is the case of Moodle, where the *business logic* code is interlined with the *user interface* code. This lack of separation-of-concerns makes very difficult any intent of adding new concerns in Moodle, due to tangling and scattering. One key lesson learned in this work is that AOSD is very valuable conceptual framework for dealing with scattered and tangled PHP web sites.

Another finding we made in this paper is that the current state-of-the-art of AOP implementations for PHP is no good enough for doing complex interventions. AOP approaches for PHP allow to define a joinpoint *before*, *around*, and *after* the call to a function. Richer *pointcut languages* are needed

for specifying a complex intervention like the one presented in this paper. As a consequence, the added functionality will be scattered throughout the program instead of being encapsulated in one *aspect*.

By definition, a *joinpoint* is a well-known point in the execution of a program. For example, it may be an object instantiation, an attribute access or a method call. Regrettably, this approach does not fit well web-specific requirements.

The main drawback in current AOP approaches in PHP is the inability to define the call to a PHP script as a *joinpoint*. When dealing with stand-alone applications, the execution flow is managed from inside the program. This is different from the Web case, where a PHP script is a "chunk of code" invoked from outside the program. There is no way of intercepting the call to a PHP script.

RELATED WORK

In this section, we analyze other initiatives relating to the integration of process-based collaboration modules in Moodle.

I. *LAMS*

LAMS stands for Learning Activity Management System [12]. LAMS has been built following a process-based approach. In this way, a LAMS *sequence* is learnflow for a unique participant. LAMS is integrated into Moodle, but the approach followed in LAMS is very different from the one presented in this paper, since LAMS is conceived as a stand-alone LMS. LAMS sequences are integrated into Moodle as a kind of resource. In this way, when a participant clicks at a LAMS sequence link, a pop-up window with the LAMS *layer* is displayed. It has to be noted that with that approach Moodle does not have the capacity to execute Moodle-native learnflows.

We identify several drawbacks in the LAMS-Moodle integration:

- LAMS only allows strict sequences, while with our approach there are many operators to specify the activities flow.
- When LAMS is integrated as a Moodle course type the GUI is LAMS, not Moodle. It is not a transparent integration.

The LAMS approach is to use the LAMS *player* to display sequences, monitor sequences, etc. LAMS remains as a stand-alone system, triggered from a Moodle course. This approach is very distinct from ours, since we add aspects to Moodle, providing Moodle with the needed functionality to sequence Moodle native activities and using the Moodle GUI. Moreover, with our approach more complex sequencing patterns are supported, since all the jBPM patterns are supported, while LAMS only supports the *strict sequence* pattern.

LAMS is an example of a *coarse-grained integration*, where the functionalities of the two systems do not overlap, since each system preserves its own GUI and its own data model.

II. IMS-LD

Some efforts to integrate IMS-LD functionality into Moodle are reported in literature [9] [10]. It seems that those plans did not reach its goal. This indicates the difficulties in updating legacy code to support learnflows: it is a very difficult work. So, special approaches based in Aspect Oriented Programming techniques are required to overcome all these difficulties.

Moodle and The Open University of The Netherlands founded a working group in June 2005. Several papers written by this working group [9] [10] address the project of integrating Moodle and IMS-LD specifications. The project had the final goal of making Moodle able to play an IMS-LD Unit of Learning (UOL). The integration process was divided into three milestones:

- To export one course to an UOL, translating the Moodle notation to IMS LD
- To import one UOL into the Content Management System and translate the IMS LD notation into a Moodle notation
- To play a UoL inside the system. Moodle stores an IMS LD information package and it runs an internal player

CONCLUSIONS

We expect that the needs of e-learning participants will evolve over time, requiring more functionality and support. Taking into account the great number of Moodle users and the great acceptance of this platform, it is important to design an evolution rather than a refactoring.

As it was stated in the introductory section, simply by putting people around a table it cannot be guaranteed that collaboration will occur. Therefore, collaboration has to be promoted in some way in order to assure its effectiveness. We propose a process-based approach to define collaboration structures (*learnflows*), similar to the approach used in the *Business Process Management* field. In this work, learnflow is provided as a *first-class object*. A learnflow can be created, instantiated, and accessed in a standardized way. Ad-hoc solutions are lacking those important features, since learnflows are harcoded with programming languages, and they are not acting as *first-class objects*, making impossible to reuse the collaboration structures. Learnflow descriptions capture the learning process and they are valuable knowledge assets, which reflect valuable know-how.

In this paper it was exposed a procedure to build *domain-specific* process machines, such as the *Learnflow Engine* one, which is learning-domain specific, on top of a generic-purpose *Workflow Engine*.

We consider that learning-domain specific processes, as well as processes belonging to other domains, can be translated into a *process lingua franca* (consisting uniquely

of nodes and transitions), and enacted with a generic-purpose Workflow Engine.

The procedure for implementing the integration code follows an Aspect-oriented approach. On top of Moodle it is placed an integration layer consisting on *aspects*. The Aspect layer controls Moodle, and delegates on the Learnflow Engine.

With this approach, all the functionalities provided by a Workflow Engine are ready to be used with learnflow purposes. In this way, the monitoring capabilities can be easily added as another *aspect*. Other capabilities are, among others, the definition of temporal points in which contents are liberated, and the invocation of external tools such as e-mail.

Furthermore, we expect this approach to be valid for integrating learnflow capabilities into other LMSs, and even into m-learning and t-learning systems.

ACKNOWLEDGMENT

This work has been partially funded by MEC grant TIN2007-68125-C02-02, and Xunta de Galicia grant PGIDIT06PXIB322285PR.

REFERENCES

[1] Moodle, http://moodle.org

[2] Victor Gonzalez-Barbone and Martin Llamas-Nistal, "Trends in Content Reuse and Standardization", *Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference*, 2007

[3] Pierre Dillenbourg, "Over-scripting CSCL: The risks of blending collaborative learning with instructional design"

[4] Davinia Hernández-Leo, Juan I. Asensio-Pérez and Yannis Dimitriadis, "Computational Representation of Collaborative Learning Flow Patterns using IMS Learning Design", *Educational Technology and Society*, 8(4), pp. 75-89

[5] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns", *Distributed and Parallel Databases*, 14(1), pp. 1573--7578

[6] Marino O. et al., "Bridging the Gap between e-learning Modeling and Delivery through the Transformation of Learnflows into Workflows", *E-Learning Networked Environments and Architectures: a Knowledge Processing Perspective*, Springer-Verlag

[7] jBPM, http://www.jboss.org/products/jbpm

[8] Isabel Michielis, "Using AOP Techniques for Dealing With Legacy Systems", *In OOPSLA workshop on Legacy Transformations: Capturing Knowledge from Legacy Systems*, Vancouver, Canada, October 2004

[9] Daniel Burgos, Colin Tattersall, Martin Dougiamas, Hubert Vogten, and Rob Koper, "Mapping IMS Learning Design and Moodle. A first understanding"

[10] Daniel Burgos, Colin Tattersall, Martin Dougiamas, Hubert Vogten, and Rob Koper, "A First Step Mapping IMS Learning Design and Moodle"

[11] NuSOAP, "http:// *sourceforge.net/projects/nusoap*"

[12] Apache Axis, "http://*ws.apache.org/axis*"

[12] LAMS, http://*www.lamsinternational.com*