Enabling Process-Based Collaboration in Moodle by Using Aspectual Services

Roberto Perez-Rodriguez¹, Manuel Caeiro-Rodriguez¹, Luis Anido-Rifon¹
¹Department of Telematics, University of Vigo
rperez@gist.det.uvigo.es, mcaeiro@det.uvigo.es, lanido@det.uvigo.es

Abstract
Moodle is the most used of open-source Learning Management Systems (LMSs). In relation with its support of Collaborative Learning, Moodle counts with a number of collaborative tools which allow participants to collaborate freely. The main drawback in current Moodle approach to collaboration is that free collaboration does not guarantee that learning is produced. To enhance the learning outcome, interactions among participants, as well as the information they interchange, have to be structured. These process-based collaboration structures are named learnflows, in analogy with the workflow domain. In this paper we propose a method to extend Moodle with learnflow capabilities, by using a generic-purpose workflow engine exposed as a Web Service to execute the learnflows, and by doing the intervention at the Moodle side in an Aspect Oriented Software Development (AOSD) fashion.

1. Introduction
In this paper, we describe a proposal for integrating jBPM workflow engine [1] in Moodle [2] following state-of-the-art techniques in application integration. 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 AOP techniques [3] (new crosscutting concerns are added in Moodle) to identify the points at which the workflow services have to be consumed.

2. Proposed solution
Figure 1 shows a diagram of the architecture as a layered system:
• 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 [5].

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 1. Architecture of the solution

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.

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. The three basic Learnflow Service Methods are:

- getPendingTasks(), which retrieves a list of the assigned tasks for a user
- endPointOfActivitySignalling(), signals the end of a task
- isUserAllowed(), check is the task is in the user’s worklist

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. 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. 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(), endPointOfActivitySignalling(), 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.

3. Conclusions

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 written 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.

4. Acknowledgements

We thank “Ministerio de Educación y Ciencia” for its support under project TIN2007-68125-C02-02, and “Consellería de Innovación e Industria” under project PGIDIT06PXIB322285PR. This research is supported by the eContentplus programme ECP 2007 EDU 417008 (www.aspect-project.org), a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

5. References