Course Generation as a Web-Service (CGWS) for E-Learning Systems *

Tianxiang Lu, Carsten Ullrich, and Babara Grabowski

German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{tianxiang.lu,carsten.ullrich}@dfki.de

Abstract. A course generator provides personalized learning experiences by assembling structured sequences of learning objects that help learners to achieve their learning goals. They can be stored in a multitude of repositories and are selected by the course generator based on a set of pedagogical methods that take into account the learners' goals and individual properties such as mastery. This pedagogical knowledge needs to be elicited from pedagogical experts and hence is rather expensive to assess. Therefore, once a course generator has been developed, it makes sense to make course generation available as a web-service for web-based e-learning environments in order to reuse the pedagogical knowledge. In this paper, we present the web-service we developed for the course generator of the LeActiveMath system. We describe the service oriented architecture during the design and the suitability via the application.

1 Introduction

Traditional web-based e-learning environments like Moodle offer access to learning resources using courses authored by humans. These courses are mostly static, hence each learner is presented with the same sequence of learning object, regardless of his individual needs. In contrast, course generation [2] allows the automatic generation of a structured sequence of learning objects that is adapted to the learners' competencies, individual variables, and learning goals. A course generator implements human expert knowledge, which is hard and expensive to assess. The contribution of this paper is to use web-service standards for providing a *Course Generator as Web Service* (CGWS). In contract to very broad approaches that define complete frameworks for educational services (e.g., [7]), this work focuses on a specific service, namely how to access the course generation knowledge in an (mostly) automatic matter. This papers presents the implementation of the CGWS (the master thesis of the first author); the underlying research is presented in [5, 6, 4]. In order to assess the potential interest and requirements of third-parties in a CGWS, we performed a survey containing several questions about potential use cases, e.g., general interest in course

* This publication was generated in the LeActiveMath project, funded under the 6th Framework Programme of the European Community (Contract Nr. IST-2003-507826). The authors are solely responsible for its content.

generation, but also technical questions, e.g, the learner models used, etc. The results of the survey served as the basis of the requirements analysis of the CGWS. Based on the requirements, we designed a loosely coupled architecture for CGWS, whose basic components we will describe in the next section. Following a top-down approach, in § 2 we define the interfaces between the server (CGWS) and the client (external web-based e-learning environments). We then illustrate the interactions between them. In the final section, we present examples of applications of the CGWS, encountered difficulties and lessons learned.

2 Design

The basic components needed for course generator are *content repositories* and *learner models*. The content repositories store the learning objects that are used by the course generator to assemble courses. However, most repositories use their own metadata schema and storage technology. Integrating data from different repositories is a well-known problem and is typically tackled by a *mediator*, which provides a uniform interface for accessing multiple heterogeneous data stores (e.g. file systems and different databases) [8]. In case of the web-service world, it is necessary that repositories can be added and removed without human intervention from the mediator side: a service should be able to register its repositories automatically. A solution to this problem is described in § 2.1.

A *learner model* [1] stores information about the learner (e.g. properties, preferences and the degree to which he has mastered the content) and hence provides information that is necessary for the personalized generation of courses. A CGWS requires that client can register its learner model, so that the CGWS can access it during course generation. The challenge for such an interface is that it has to cope with various types of learner models, but also the fact that some web-based e-learning environments do not even have a learner model.

A *course generator* uses the information provided by the above components and assembles learning objects from one or many repositories into a personalized course with respect to a pedagogical learning goal (a task, [6]). Tasks are fundamental concepts in course generation. A pedagogical task represents the learning goal of a user and consists of a *pedagogical objective* and of a *set of learning objects* that specifies the course's target concepts. An example is the task *(discover (def_slope))*, which represents the learning goal of a learner who wants to discover and understand the mathematical definition of the "slope". A different learning goal is, say, *(train (def_slope))*. A course supporting the user in reaching the former learning should contain all learning objects required by this individual user to understand the goal concept.

2.1 Interfaces of the CGWS

The CGWS provides two main kinds of interfaces: the core interface and the repository integration.

The core interface consists of the following methods: *getTaskDefinitions()* is used to retrieve the pedagogical tasks which the course generator can process (described in XML format); *generateCourse(objective, learning object _Ids, learnerId)* starts the course generation on the given educational objective, identifiers of learning objects and the identifier of learner (or LearnerKnowledgeMap, see § 2.3). The result of the course generator is a structured sequence of learning objects. The format of the result complies to IMS-CP, an established standard for the exchange of courses between different web-based e-learning environments.

The interface for repository registration consists of the following methods: *getMetadataOntology()* provides the client with a description of the metadata structure used in server. The description is provided as an ontology that describes the pedagogical types and relationships of learning objects [5].

The method *registerRepository(name, rep_url, ontology_url, ontology_map_url, testLO)* registers the repository that the web-based e-learning environments client wants the course generator to use. Since different repositories often use different metadata for describing the learning objects they store, the client needs to provide information to the CGWS that helps to "understand" the metadata. Therefore, the clients specifies the URL of the ontology describing the metadata structure used in the repository and in addition an URL of an ontology mapping between the two ontologies. This way, the mediator can translate between the metadata used in the server and in the client.

The method *unregisterRepository(rep_url)* unregisters the given repository.

2.2 Client Interfaces

A web-based e-learning environments client needs to provide the following interfaces: the *ContentAPI* needs to be able to answer queries coming from mediator that ask for (a) the type of the given learning object; (b) all the learning objects that are connected to a given learning object by a given relation; (c) all properties the given learning object has [4].

The *LearnerPropertyAPI* makes the learners' properties accessible to the CGWS in case the client contains a learner model and wants the course generator to use it. In the current implementation, this interface is not yet implemented. It would require a mediator architecture similar to the one used for repository integration.

2.3 Interaction between Client and Server

In this section we describe the communication between client and server. We will focus on the most relevant interactions.

Repository Registration The registration phrase happens as follows: In a first step, the client retrieves the metadata ontology used in the CGWS. The ontology is then used to generate a mapping between the metadata used on the server and the one on the client. In the current system, the existing mappings were manually authored. Then, the repository is registered using the

method *registerRepository()*. The repository is added to the list of available repositories and made known to the mediator. Subsequently, the mediator fetches the ontology mapping from the client and generates a wrapper for querying the *contentAPI* of the client.

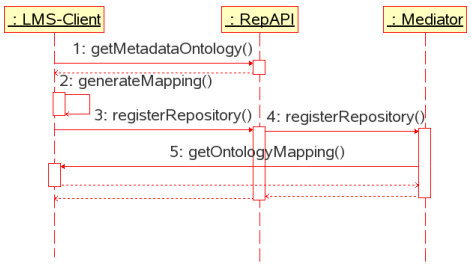

Fig. 1. web-based e-learning environments client registers its repository into CGWS

The most important step in the registration and the contribution of this work is the automated generating of a wrapper (adapter or proxy), which takes charge of binding and querying the *contentAPI* during the course generation. The *contentAPI* should be provided as a web-service, so that the communication can be performed automatically. In this way, a repository can be registered automatically during run-time, without human implementation work required for the wrapper.

Generating a course A client starts the course generation using the service method *generateCourse()*. During the generation, the course generator queries the mediator for the learning objects needed for the course but also the learner model for information about the learner. After the course was generated, the course generator will return an internal data structure that represents the structure of the course. This structure is transformed to an IMS manifest, packaged in a SOAP message and send to the client.

Often, web-based e-learning environments do not have a learner model, but still want to generate adaptive courses for their users. For these cases, the course generator allows to define a temporary learner model, which consists of property-value pairs that represent the current knowledge of the user. This model is called *LearnerKnowledgeMap*. For instance, a first year university student who masters Differential Calculus can have the following *LearnerKnowledgeMap*: ((”‘Definition of Differential Calculus, good”’), (”‘Educational level, University, 1. Semester”’)). A client can use a *LearnerKnowledgeMap* by calling the service method *generateCourseWithLearnerKnowledgeMap()*. If the course generator requires the value of a property not stored in the map, a default value is used.

3 Applications

The design was implemented using Java and Axis2. The logic part on the server side is implemented as Java packages, which call the course generator directly using Java-API or using XML-RPC if the server is distributed. Axis2 is responsible to facilitate the web-service details, which includes the WSDL generation and the hot deployment of web-service. SOAP is used for message exchange.

The course generator described in this paper was developed in the FP6 project LeActiveMath, in a joint cooperation between AI and pedagogical experts. Figure 2 contains a screenshot of a course generated in LeActiveMath.

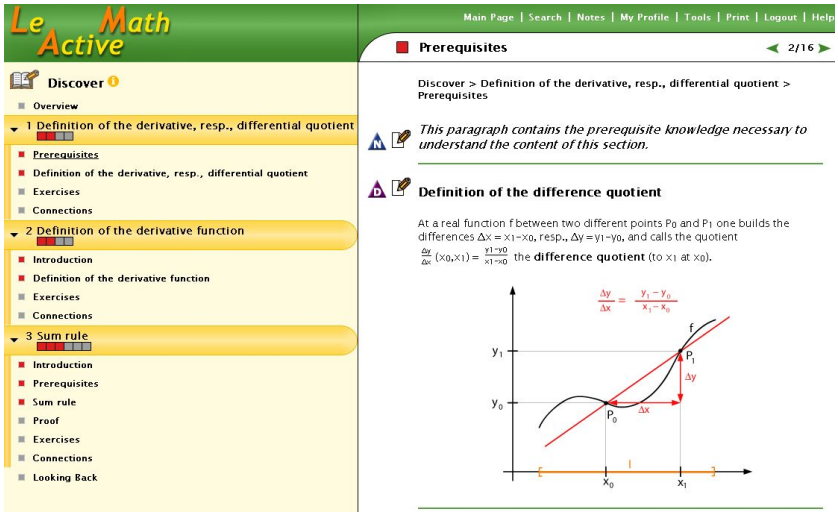

Fig. 2. A generated book in ActiveMath

An external client of course generator was developed at the University of Applied Sciences Saarland. The MathCoach system is a web-based e-learning environments tool especially designed for exercises and experiments within a mathematical context [3]. Using the mechanisms described above (e.g., an ontology mapping), it was possible to make the functionalities of the course generator available to MathCoach.

A third use case of remote access to the course generator was done in the teal project, which targets e-learning in office environments. This illustrates the general applicability of our work. Currently we are working on an integration of the CGWS into the web-based e-learning environments of the distant university of Shanghai Jiao Tong University.

During the application we encountered several difficulties: (a) while the mediator architecture allows an easy technical integration, it is still a challenge

to design the ontology mapping, especially in the often occurring case of poor metadata; (b) for technically unexperienced clients the usage of Web services is still challenging despite being a standardized technique today.

4 Conclusion and outlook

In this paper we described a service oriented architecture for course generator implemented as a web-service. This allows external clients to access the services provided by the course generator, a component that substantially relies on pedagogical knowledge and therefore is expensive to design. This work is the first approach to provide such a *course generator as web-service* (CGWS) to external web-based e-learning environments, which improves the remote communication between systems or machines and enables a cooperated environment of developing web-based e-learning environments. Additional research needs to be invested regarding the generic integration of learner models. In the near future, we will provide an official web-service access point for CGWS so that other web-based e-learning environments can reuse the course generator.

References

1. Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. *The Adaptive Web: Methods and Strategies of Web Personalization*, 2007.
2. Peter Brusilovsky and Julita Vassileva. Course sequencing techniques for large-scale webbased education. *International Journal of Continuing Engineering Education and Lifelong Learning*, 13(1/2):75–94, 2003.
3. Barbara Grabowski, Susanne Gäng, Jörg Herter, and Thomas Köppen. MathCoach and LaplaceScript: Advanced Exercise Programming for Mathematics with Dynamic Help Generation. In *International Conference on Interactive Computer Aided Learning ICL*, Vilach, Austria, 2005.
4. Philipp Kärger, Carsten Ullrich, and Erica Melis. Integrating learning object repositories using a mediator architecture. In *Proceedings of the First European Conference on Technology Enhanced Learning*, pages 185–197, Heraklion, Greece, oct 2006. Springer.
5. C. Ullrich. Description of an Instructional Ontology and its Application in Web Services for Education. In *Proceedings of Workshop on Applications of Semantic Web Technologies for E-learning, SW-EL'04*, pages 17–23, Hiroshima, Japan, 2004.
6. C. Ullrich. Course generation based on HTN Planning. In A. Jedlitschka and B. Brandherm, editors, *Proceedings of 13th Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems*, pages 74–79, 2005.
7. Peter Westerkamp. *Flexible Elearning Platforms: A Service-Oriented Approach.* Logos Verlag, Berlin, 2005.
8. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *The IEEE Computer Magazine*, 1992.