ReST-Based Web Access to Learning Design Services

Juan Manuel Dodero and Ernie Ghiglione

Abstract—The IMS Learning Design model defines how users are engaged in learning activities to use learning objects and learning services. Although these are similarly accessible from URLs in a learning design, they are essentially different in the form of accessing their features and operations from a learning activity user interface. This paper proposes a Representational State Transfer (ReST) architectural style of accessing a learning service and its constituent resources from a Web-based environment. A methodology to guide the design of learning service access is also described. The method is applied to integrating resources from a Web-based environment. A methodology to guide the design of activity user interface. This paper proposes a Representational State Transfer different in the form of accessing their features and operations from a learning activity and validated on actual wiki services providing ReST-based programming interfaces.

Index Terms—Computer-assisted instruction, Web-based services, software architecture patterns, hypermedia architectures.

1 INTRODUCTION

A Learning Design (LD) defines a set of people in specific groups and roles engaged in learning activities with a number of resources and services. The IMS LD specification provides a framework of elements, such as learning objects and learning services, which can be used to formally describe an LD. What distinguishes a learning object from a learning service is that for the former, its location or Uniform Resource Identifier (URI) is known at design time, whereas services are enacted after the LD is instantiated under the form of a Unit of Learning (UoL) [1]. Learning objects and learning services are essentially different at the time when they are available to the learning designer. But, from an accessibility viewpoint, learning objects and learning services are also different on their interfaces, which are the way to access and exploit those features that are useful for the learner.

A learning service is a functional concept supporting a user in the learning process [2]. In the IMS LD model, a learning activity and its services are mediated by an environment, as depicted in Fig. 1 [3].

Typical learning services include authorization and authentication, mail and messaging, conferencing, search, sequencing, and collaborative editing and assessment, among others. These usually provide an Application Programming Interface (API) to enable any external application, such as LD players, to send requests to and receive answers from the service in a Web-based runtime environment. IMS LD-based learning services require a Web Service Description Language (WSDL) specification of the functions the service provides [2].

The focus of this paper is on how a learning service is accessed from a particular LD activity on a Web-based environment. The main issue in achieving interoperability between IMS LD-compliant players and learning services is having a generic and comprehensive description of any feasible service for teaching and learning. Current IMS LD interoperability solutions such as the ELF e-learning framework[1] are focused on providing a tightly coupled, function-based access interface to a model of the service; this model often is not exposed, and it is assumed that it is shared by all learning activities that consume the learning service. However, it is often sufficient to know what resources and features are available in the service model and how to access them, while keeping the interaction between activities and services as simple as possible.

Learning services can be classified according to the kind of service provider and to the service containment degree, which means that you need to locally download the service software to be able to access it (tight containment), or you need to set up a remote access to the service before using it (loose containment). On the one hand, the provider can be a standalone service or an LMS-hosted service. Instances of standalone services include all kinds of Web 2.0 applications—see, for example, Google maps used as a learning service in [4]—whereas LMS-hosted services usually come under the form of LMS-specific activities (e.g., moodle or dotLRN assignments, questionnaires, forums, wikis, and so forth), LMS-hosted services are more difficult to access since they only exist in the context of an LMS-specific course. On the other hand, learning service access is classified according to the containment degree. We focus our work on accessing loosely contained, standalone services, since 1) the transformation of an LMS-hosted service to the standalone version is a technical issue that can be easily managed by the LMS; and 2) tightly contained Web services require first downloading the access software, thus any accessibility issue can be readily solved.

The key issue is providing learning services with an easy and flexible approach to access them from learning process activities when these are run—possibly on an external LD sequencing service. The access requirements can be summarized as follows:

- Open integration: Services must publish their interfaces so that their operations or primitives can be called from external activities.
- Fine grain control: A fine grain access level must be provided to exploit the service interface primitives on an individual basis, as well as to access the back-end objects or resources supporting those operations.
- User interface decoupling: The service primitives must be decoupled from the user interface used to call them, so that it permits an easy replacement of either the interface or the service.

2 ACCESSING LEARNING DESIGN SERVICES

From the very beginning of the IMS LD specification, Olivier and Tattersall [1] argue that the key issue to be addressed is how to add services to an LD such that they still retain portability across different LD-compliant platforms. The approach to this issue taken by the SLeD and CopperCore Service Integration (CCSI) is to provide generic descriptions of operations and methods for each service without requiring the service provider to update their API to fit with a particular description [2]. Such descriptions are usually provided by a WSDL specification, which has to be updated if the service interface changes or is extended. Moreover, if the particular service provider has an operation that does not fit in the generic operations, CCSI provides an adapter to take advantage of these operations. However, you need to update the player, making it difficult to add a new service type [5].

The LAMS approach took the other way around and suggested that, in order to create tools and services that are meaningful from an LD perspective, it is more practical to build them from scratch than reengineer existing ones [6]. The issue here is the loss of
CRUD function on the resource identified by id. Excepting the POST request, the verb is omitted since it is implicit in the HTTP method. The representation format of the output (HTML, XML, and so forth) is part of the HTTP header and can be specified on the URI as /resource/id.format. Following ReST principles ensures an open integration API to access Web services or resources, as required above. Unlike a SOAP service, where all requests are directed to a single endpoint, a ReSTful Web service accepts requests at multiple URIs—each mapped to a different resource. Instead of requiring a WSDL to describe all of the methods and arguments the LD might call, the specification of a ReSTful service is just the list of resources it exposes. This characteristic fosters the fine grain control requirement mentioned above. The ReST-based style of accessing learning services provides a loosely coupled integration between LD activities and learning service providers, so endorsing the user interface decoupling requirement.

2.2 ReST-Based Learning Service Access

The ReST-style access to learning services consists of a façade object [12], which is represented by a URL and receives requests to create or manipulate an object instance, which in turn can be referenced using a particular URI. Such objects represent resources that are inbounds to the learning service façade and are referenced by arguments of the service operation, as depicted in Fig. 2.

There are several benefits of the access to learning services derived from the ReST architectural style:

- **Uniform access method:** The main advantage is the uniform interface by which all resources are accessed with a generic set of HTTP methods.
- **Addressable resources:** The service implementation is comprised of resources that are named and addressable from the service provider outbounds.
- **Resource interconnection:** The representations of resources are interconnected using URLs, thereby enabling an LD to progress from one state/activity to another in a navigational style.
- **Extensibility:** ReST provides a flexible scheme to easily implement changes and extensions to services and service wrappers.

The architecture of how ReST-based services are accessed describes layered mediators or wrappers, such as proxy servers, cache servers, gateways, and so forth, that can be inserted between the LD player and the service provider to support performance, security, and so forth. In fact, the generic service integration approach [9], which interposes a wrapper for service execution enactment and awareness, can be readily combined with a ReST-based architecture by deploying the wrapper as a kind of layered component.

2.3 Service Access Design Methodology

Weller et al. [5] enacted the need of a methodology for describing learning services so that new ones can be added to an LD. The methodology must encompass the means by which services are described and how LD recognizes these. Under the ReST principles, we have taken the following steps for the proposed design methodology:

1. Define the façade of the service. This can be a simple interface—whether the correspondence from service primitives to their object implementation is simple—or an adapter component, which translates service requests to the object collaboration that realizes them. For instance, the operation create_post(user, post_contents) used to create a post in a blog service can be implemented by a single method call to user.create_post (post_contents) on the user.
object. However, the operation get_user_tagged_posts (user, tag), which retrieves the postings of a user that are marked with a specific tag, might require an adaptation to translate it into a more complex call, such as: tag.filter (user.get_posts()); now, it requires knowing not only the operation interfaces but also the relationships among user, post, and tag objects. The façade is not really indispensable, but it is a procedure-oriented gateway to access the resource-oriented ReST service, required if the LD player needs a typical function-based API.

2. Design the object model of the service resources. The remote resources involved in a learning service invocation usually exhibit relationships among them, such that chains of objects and relationships are also referenced in the call. For that aim, an object model of these service resources must be defined. In the blog service example, user objects are \(1:n\) associated to posts, tags are \(n:m\) associated to posts, and users are \(n:m\) associated to tags.

3. Map service operations to ReSTful URIs. Once the service interface and the component objects are defined, the URI scheme is defined. This scheme is typified by ReST as a chain of \(\text{object_type/object_id}/\) parts, optionally ended by a \(\text{/request}\) part. For example, the URI \(\text{/users/x/posts/y/}\) might represent the posting \(y\) of user \(x\) in a blog service; and \(\text{/users/x/posts/new}\) would represent a request to create a new post for user \(x\).

4. Provide a written specification of the ReST service interface. This step is not mandatory but allows for summarizing the permissible URIs designed in the previous step, by means of an explicit description in Web Application Description Language (WADL). It works similarly to WSDL with SOAP and essentially enables the automatic generation of a user interface to access the service.

5. Map ReSTful URIs and invocation arguments to the LD user interface or interaction elements. The LD activity invokes the service from a user interface element. Availability of these user interface arguments depends on the kind of LD player invoking the service. On the one hand, the IMS LD specification uses properties to retain all values from user interaction. Property values are set and viewed through global elements, which are IMS LD instructions that can get and set property values, usually contained within forms delivered to the user. Then, global elements are mapped to URIs. On the other hand, LAMS LD activities directly use HTML forms and controls to handle the interaction, so these would be mapped to the service URLs. It is worth mentioning that the optional step 4 would considerably facilitate this task, since the mapping can be automated by generating the user interface from the WADL specification.

3 VALIDATION CASE: HOW TO ACCESS WIKI LEARNING SERVICES

Wikis are readily available Web applications that provide a kind of collaborative authoring service, which can be usefully integrated in a learning process as long as an open API is provided by the wiki. The case of wikis is used to show how the above-mentioned benefits derived from ReST apply throughout the steps of the design methodology.

3.1 Wiki Service Façade

We have applied the design method described above for a set of basic wiki services. Fig. 3 depicts the wiki service API, which consists of a set of operations to be invoked from a running LD. This is a simple façade of the wiki service (methodology step 1) because actual wikis provide additional functionalities on pages for tagging, watching, access control, and so forth. Nevertheless, the service interface can be extended if needed, just by further applying the design method on the extended functionalities.

```
<?xml version="1.0"?>
<interface>
  <service name="WikiService">
    <operation name="get_pages" parameter="User">Page</operation>
    <operation name="get_page_versions" parameter="Page">PageVersion</operation>
    <operation name="get_page" parameter="User,pagoid" parameter="String">Page</operation>
    <operation name="get_page_version" parameter="Page,pagoid,version" parameter="String">PageVersion</operation>
    <operation name="create_page" parameter="User,pageContents">Page</operation>
    <operation name="update_page" parameter="Page">void</operation>
    <operation name="restore_page_version" parameter="Page,pageVersion">Page</operation>
    <operation name="remove_page" parameter="Page">void</operation>
  </service>
</interface>
```

Fig. 3. UML façade of the wiki service.
3.2 Wiki Object Model

Although the wiki API only provides a functional abstraction of the service operations, there exists an underlying object model of resources available through these operations. For instance, the operation create_page (user, page_contents) is called to create a page associated to a contributing user; hence, it deals with two kinds of objects, i.e., users and pages. According to the methodology step 2, the object model in Fig. 4 represents the object types involved in the wiki service interface.

3.3 Wiki ReSTful API

The ReST API that results from mapping the service operation to ReSTful URIs (methodology step 3) is described in Table 1. ReSTful URIs provide a simple and standard syntax to describe User-Page and Page-PageVersion relationships of the object model, such that any operation invocation can be easily generated from that model as an HTTP method call. For example, the get_page_versions (page_id) operation retrieves all versions of a given page; to implement a ReSTful call to this service, the URI /pages/page_id/versions is formed; parts of the URI prefixed with a colon stand for the actual values of the operation arguments. Therefore, all resources of the wiki (i.e., users, pages, and page versions) can be uniformly named, accessed, and addressed out of the service invocation is specified by appending ”.xml,” ”.html,” or any other desired suffix to the URI. For instance, /pages/32/ the service invocation is specified by appending ”.xml,” ”.html,” or any other desired suffix to the URI. For instance, /pages/32/. The advantage of ReST style is that the format to deliver the output of any operation invocation can be easily generated from that model as an HTTP method call. For example, the get_page_versions (page_id) operation retrieves all versions of a given page; to implement a ReSTful call to this service, the URI /pages/page_id/versions is formed; parts of the URI prefixed with a colon stand for the actual values of the operation arguments. Therefore, all resources of the wiki (i.e., users, pages, and page versions) can be uniformly named, accessed, and addressed out of the service provider through common HTTP requests.

One advantage of this approach is that it provides a fine-grained access to the resources that compound the wiki service. Instead of having a single coarse-grained URL to access the overall service, component objects of the service (i.e., users, pages, and page versions) are visible and can be navigated from the LD. Another advantage of ReST style is that the format to deliver the output of the service invocation is specified by appending ”.xml,” ”.html,” or any other desired suffix to the URI. For instance, /pages/32/ edit.iphone would deliver a form to edit the page with id 32 on an iPhone. This enables the interconnection of resources required to navigate from one LD activity to another, and it can do it for a diversity of delivery formats and user devices.

3.4 Integrating URIs in the LD Activity Interface

As the last step of the methodology, we realized an exploratory evaluation of how the ReST-based scheme is integrated on an LD activity interface for three wiki services, namely MindTouch DekiWiki,2 OpenMaru SpringNote,3 and Google JotSpot.4 These have been selected since they provide an open API to operate with the set of available resource objects. Some of these wiki applications do not provide a completely ReSTful API. For example, JotSpot requires a POST removePage command instead of using the ReST-compliant form of deleting a resource by means of the HTTP DELETE message. However, they all provide the required ReST functions as verbs to access the essential wiki services.

Resources available through the API are different for each actual wiki application, as shown in Table 2. The first column summarizes the resource objects that might be exposed through the wiki API, while the rest of the columns indicate if they are available (×) or not (−) on each wiki service. Some do not offer their complete API through ReST. For example, JotSpot API can handle tags and attachments, but its reduced ReST version does not include such resource types. In such cases, they are marked with a dot. Although resource objects have methods that implement the service operations, these have not been described as part of the table.

Then, we explored how wiki services are accessed from an LD activity. The activity client interface has been developed with Google Web Toolkit (GWT). The following code listing shows how the get_page (page_id) service operation is implemented to enable retrieving a wiki page from the activity:

```java
class PageResource implements Resource {
    //
    public void get_page(int page_id) {
        //…
    }
}
```

The program code shows how different service providers can be integrated. First, the constant SERV_PROVIDER is configured with the hostname of the wiki application provider (i.e., DekiWiki, SpringNote, or JotSpot). Second, GET_PAGE_CMD holds the ReST-style command for retrieving the pages. According to ReST principles, it should be the string ”pages/” but it can be configured for any provider that does not fully comply with ReST vocabulary (e.g., listAll retrieves all pages in JotSpot). The rest of the details of the request (e.g., the page id) are uniformly coded in the method ResourceCollection.find. Therefore, adapting the client access interface from the LD activity to the learning service is as straightforward as configuring the URI access point to the resource and the ReST-based verb vocabulary. This way, ReST facilitates

---
updating the service access interface due to an eventual change or extension to the service.

## 4 CONCLUSIONS AND FUTURE WORK

This paper has described a ReST-based architectural style of accessing learning services from LD activities on a Web-based environment and has provided a methodology for that aim. The proposed ReST solution is simple, since it is based on few principles and well-defined operations, scalable, since it relies on stateless protocols and distributed state, and layered, allowing for any number of intermediaries between activities and services. The service access requirements of open integration, fine grain control, and user interface decoupling are fulfilled by the architectural style and design methodology.

Providing activities with a user interface to access ReST-based learning services is still a matter of configuration. In further work, this interface might be generated from a WADL service specification provided beforehand. Another line of work is integrating the ReST-based service access on a layered wrapper that enacts the service and multiplexes service calls. This would provide the generic service access approach [9] an improved scalable design. In this work, we have only considered the integration of service-specific primitive operations within LD systems. The actual list of functionalities that need to be integrated must cover the complete life cycle of the service, from authoring and enactment to monitoring to interoperability facilities. This is an ongoing effort that has been undertaken in the FLEXO project, funded by the Spanish Ministry of Industry and Commerce, and realized in cooperation with several Spanish companies and universities and the LAMS Foundation.

### ACKNOWLEDGMENTS

This work was supported in part by the “Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica,” Projects TSI-020301-2008-19 and TSI-020501-2008-53.

### REFERENCES


[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.