SMashup Personal Learning Environments

Mohamed Amine Chatti[1], Matthias Jarke[1], Zhaohui Wang[1], and Marcus Specht[2]

[1] Informatik 5 (Information Systems), RWTH Aachen University
{fchatti,jarke,wangg}@dbis.rwth-aachen.de,
[2] Open University of the Netherlands, Netherlands
marcus.specht@ou.nl

Abstract. Mashups have become the driving force behind the development of Personal Learning Environments (PLE). Creating mashups in an ad hoc manner is, however, for end users with little or no programming background not an easy task. In this paper, we leverage the possibility to use Semantic Mashups (SMashups) for a scalable approach to creating mashups. We present the conceptual and technical details of PLEF-Ext as a flexible framework for mashup-driven end-user development of PLEs. PLEF-Ext uses the Service Mapping Description (SMD) approach to adding semantic annotations to RESTful Web services, and leverages the SMD annotations to facilitate the automatic data mediation and user-friendly creation of learning mashups.

1 Introduction

Recently, the mashup concept has emerged as a core technology of the Web 2.0 and social software movement. A mashup indicates a way to create new (Web) applications by combining existing data and services from several sources. In the past few years, Web mashups have become a popular approach towards creating a new generation of customizable Web applications. The popularity of Web mashups has mainly been driven by the increasing popularity of lightweight RESTful Web services, AJAX, and JSON that build core technologies in the Web 2.0 movement. Mashup development has also become a very popular re- search topic in TEL. Many TEL researchers recognized the value of mashups and have already adopted the mashup concept for Personal Learning Environment (PLE) development. The idea behind mashup PLEs is to let learners create their very own learning mashups that leverage components and content generated by learning service providers and other learners around the Web. Yet, most of the proposed mashup PLE approaches are based on widgets. In this paper, we go a step further toward supporting the mashup of RESTful services and the use of semantic approaches to deal with service integration and mediation within mashup PLEs. We propose a mashup PLE framework, called PLEF-Ext, that enables a user-friendly creation, management, sharing, and reuse of learning mashups.

The paper proceeds as follows. In Section 2, we discuss how mashups can provide a powerful tool to build PLEs. In Section 3, we provide an overview of some popular and representative mashup development tools and frameworks, and summarize the main problems in mashup development today, focusing specifically on the data mediation and integration challenges. In Section 4, we highlight the benefits of using semantic mashups for a scalable and flexible mashup development. Section 5 presents the Semantic Mapping Description (SMD) approach to adding semantic annotations to RESTful Web services and shows the benefits of adopting a semantic mashup approach based on SMD. We follow in Section 6 with the conceptual and technical details of PLEF-Ext, a mashup PLE frame- work that can help learners share, find, integrate, reuse, and easily remix learning services based on a semantic description of the same. And finally, we summarize our findings in Section 7.

2 Mashup Personal Learning Environments

A Personal Learning Environment (PLE) is a learner's gate to knowledge. From a technical point of view, a PLE can be viewed as a self-defined collection of services, tools, and devices that help learners build their Personal Knowledge Networks (PKN), encompassing tacit knowledge nodes (i.e. people) and explicit knowledge nodes (i.e. information). Thus, mechanisms that support learners in building their PLEs become crucial. Mashups provide an interesting solution to developing PLEs. We differentiate between two types of mashups:

- **Mashups by aggregation** simply assemble sets of information from different sources side by side within a single interface. Mashups by aggregation do not require advanced programming skills and are often a matter of cutting and pasting from one site to another. Personalized start pages, which are individualized assemblages of feeds and widgets, fall into this category.
- **Mashups by integration** create more complex applications that integrate different application programming interfaces (APIs) in order to combine data from different sources. Unlike mashups by aggregation, the development of mashups by integration needs considerable programming expertise.

Most of the state-of-the-art mashup PLE solutions only focus on the first type of mashups, i.e. mashups by aggregation. These solutions mainly support learners in juxtaposing content from different sources (mainly feeds and widgets) into a single interface. Examples include PLEF[1], MUPPLE[2], and - although not designed as educational technology - Personalized Start Pages such as iGoogle[3] and Netvibes[4]. In the next section, we discuss the current status of mashup development with a focus on the second type of mashups, i.e. mashups by integration.

3 Mashup Development

The Internet and its related technologies have created an interconnected world in which we can easily exchange and reuse information in unforeseen, unexpected ways. Web services are emerging as a major technology for deploying automated interactions between distributed and heterogeneous applications [2]. In the Web 2.0, services based on the representational state transfer (REST) paradigm [3] are increasingly popular for Web development. RESTful services often take the form of RSS/Atom feeds and AJAX based lightweight services, which explains their greater success compared to heavyweight services, which are based on the Web Services Description Language (WSDL) and SOAP. The output formats of RESTful services, often in the form of Extensible Markup Language (XML) or the more lightweight form JavaScript Object Notation (JSON), make RESTful services ideal for AJAX-based mashups.

Developing mashups is however not an easy task. In general, mashup development is still very much an ad hoc activity. Mashups are often generated manually using normal Web development technologies such as HTML, CSS and JavaScript [4]. Creating a mashup in a manual manner is a very time consuming task and impossible for the typical Web user [5]. A key difficulty in creating mashups is data mediation between the services to be mashed up. Typically, in order to create a mashup, the user would need to understand not only how to write code but also the APIs and descriptions of data formats of all the services that need to be included in the mashup [6].

To lower the barrier of creating a Web mashup, leading companies are now actively developing different mashup building tools and platforms, that require little to no programming knowledge from the user. Yahoo! Pipes[5], Microsofts Popfly[6] and Google Mashup Editor[7] are some well-known examples of mashup platforms that users have largely adopted. In general, these platforms provide a higher level of abstraction and use visual programming techniques to facilitate the creation of mashups. For instance, Yahoo! Pipes provides a visual editor to wire different graphical elements to each other, based on different kinds of data processing operations such as regular expressions,

---

1 http://eiche.informatik.rwth-aachen.de:3333/PLEF/index.jsp

2 http://mupple.org

3 http://www.google.com/ig

4 http://www.netvibes.com/

5 http://pipes.yahoo.com/

6 http://www.popfly.com/

7 http://code.google.com/gme/

filters, sorting or looping instructions. Similarly, Microsoft Popfly provides a Web-based GUI, where the user can connect different Web services to each other by dragging the blocks representing them into the main display and drawing wires to connect those services. Google Mashup Editor (GME) provides a higher level JavaScript API for manipulating data programmatically. A detailed description and comparison of these and other mashup development tools is provided in [7].

Google Mashup Editor has not been designed to be used by end users without background in programming. It is more suitable for amateur and professional mashup programmers. And, other mashup editors, such as Yahoo! Pipes and Microsoft Popfly, that are intended for non-professional mashup developers suffer from two main limitations.

First, most of the mashup tools are restricted to services that have standard types of outputs, such as RSS or ATOM. These mashup tools are particularly capable at processing and creating new feeds. A common feature is (1) pull data from external RSS and Atom feeds, (2) filter, sort, and combine many feeds into one, (3) grab the output as RSS, JSON, KML, and other formats, and (4) include the result as a widget in other Web sites.

Second, the usage scope of these mashups tools is often limited to services that are internal to the company where the mashup tool was developed (Yahoo! Pipes, for example, adds Yahoo! Maps to any pipe containing GeoData). And, in most cases these mashup tools are restricted to a set of few of popular services such as Flickr, Digg, or popular map, traffic or weather services.

As a consequence, a plethora of existing (RESTful) services cannot be used by these mashup tools. If one of the companies behind the aforementioned mashup tools, for some reason, would be willing to add a new external service to their supported service pool, it would be necessary to implement a new wrapper in order to accommodate the new service. This is, however, not a scalable solution due to the rate at which new services are coming online [6]. Moreover, it would be extremely difficult to work with a service whose inputs and outputs are in a different format. This makes it difficult to address issues related to data interoperability, integration, and mediation [5].

The concept of Semantic Mashups (SMashups) [8] addresses these limitations by proposing the semantic annotation of Web services as a solution to the service integration and mediation challenges.

4 Semantic Mashups (SMashups)

The challenges of service interoperability, reuse, integration, and mediation have led to several proposals for Semantic Web services. Several research projects have looked at semantics for traditional (WSDL or SOAP) Web services to help address heterogeneity, reuse, and mediation challenges, and the community took a step toward supporting semantics for Web services by adopting Semantic Annotation for WSDL (SAWSDL)[8] as a W3C recommendation in 2007. The SAWSDL specification enables semantic annotations for Web services using and building on the existing extensibility framework of WSDL [2, 9]. In SAWSDL, se- mantic annotations describing inputs, outputs, operation, interfaces, and faults, are embedded as properties in the WSDL [5].

Recently, driven primarily by the popularity of lightweight RESTful Web services, AJAX, and JSON that build core technologies in the Web 2.0 movement, attention has shifted to using semantics to annotate RESTful Services. For instance, [5, 6] proposed a framework called Semantic Annotation of REST (SA-REST) to add semantics to RESTful services. SA-REST builds upon the authors' original ideas in WSDL-S [10], which was the primary input of the W3C recommendation SAWSDL.

The key difference between SAWSDL and SA-REST is that unlike SAWSDL, where annotations are

[8] http://www.w3.org/2002/ws/sawsdl/

added directly in the WSDL of the service, SA-REST annotations have to be embedded into the HTML page that describes the service. Consequently, SA-REST uses microformat-based approaches such as RDFa[9] and Gleaning Resource Descriptions from Dialects of Languages (GRDDL)[10] to add and capture annotations via properties of HTML elements [6].

The major drawbacks of an SA-REST approach to add semantic annotations to RESTful services are threefold:

First, semantic annotation with SA-REST requires the use of microformats. Microformats, however, come in many different competing forms. And, even the process of adding semantics with standardized microformats such as RDFa or GRDDL is not straightforward. For instance, to annotate a HTML page with GRDDL the service provider needs (1) to embed the annotations in any, not necessarily standardized, microformat, (2) add the URL of the GRDDL profile to the head element in the HTML document, and (3) add to the head element a link tag that contains the URL of an XSLT that translates the GRDDL profile into SA-REST specific RDF triples [5]. The complexity of these microformats makes it hard for SA-REST to be adopted by service providers for the semantic annotation of their RESTful services.

Second, the annotation of a concept (e.g. input, output, operation) in SA-REST is a way to tie together the concept to a class that exists in an ontology [5]. SA-REST, thus, presumes an agreement among all service providers on a common ontology. This is however not realistic, since it would be impossible to get all service providers to agree on a single ontology to describe their concepts.

Third, besides linking a concept to an ontology class, data mediation in SA-REST requires the specification of a lifting and lowering schemas, that is mapping of the data structure that represents the input and output of a given service to the data structure of the ontology. The use of lifting and lowering schemas means, on the one hand, extra complicated work for the service provider to work with XSLTs or XQueries to specify and keep up-to-date its lifting and lowering schemas, and, on the other hand, higher server workload and additional server-side code in order to deal with parsing and applying the lifting and lowering mappings between the services to be mashed up and the ontology.

Recently, Kris Zyp [11] has proposed a more lightweight and standards-based approach to adding semantics to RESTful services, called Semantic Mapping Description (SMD).

5 Service Mapping Description (SMD)

In this section, we discuss the Service Mapping Description (SMD) approach to adding semantic annotations to RESTful Web services. SMD is a flexible and simple JSON representation describing Web services. A wide array of web services can be described with SMD including RESTful services and JSON-RPC services. SMD uses JSON Schema to provide a definition for a variety of available services, and document how to call the services, what parameters are expected, and what to expect in return [11]. Figure 1 shows the Yahoo! search service semantically annotated with SMD.

Similar to SA-REST, SMD facilitates data mediation between the services to be mashed up, since it provides a thorough description of how a service can be invoked, the type of data that should be passed to the service and what will be returned from it. The main advantage of SMD, as compared to SA-REST, is that it is based on JSON. JSON is a lightweight data interchange format whose simplicity has resulted in widespread use among Web developers. JSON offers two major advantages over XML. First, JSON-encoded data is less verbose than the equivalent data in XML. XML uses duplicate start and end tags to wrap data values. A JSON object, by contrast, is simply a series of comma-separated name:value pairs wrapped in curly braces. This yields better performance of the JSON-based Web

[9] http://www.w3.org/TR/xhtml-rdfa-primer/

[10] http://www.w3.org/TR/grddl/

applications because JSON data downloads more quickly than XML data.

JSON's second benefit is that it's easy to parse using any programming language, and its structures map more directly onto the data structures used in modern programming languages. Because it is essentially a string representation of a JavaScript object (hence the name), browsers can parse JSON data simply by calling the JavaScript "eval" function. Any field of the JSON object can then be directly accessed by name.

Fig. 1. Semantic Annotation of RESTful Web Services with SMD

JSON is also the basis of a very powerful technique for building client-based (in-browser) mashups, namely JSON with Padding (JSONP), which was created as a workaround to the Same-Origin Policy (SOP) problem. In Web 2.0, Asynchronous JavaScript and XML (AJAX) is becoming the driving force behind mashups. In AJAX, data is retrieved using the XMLHttpRequest function, which is an API that lets client-side JavaScript make HTTP connections to a server. The SOP, however, prevents the AJAX code from making an XMLHttpRequest call to a URL that is not on the same domain as the current page, because of the cross-domain attacks that could result.

The normal way around SOP is to have the Web server behave as a proxy. The Web page then requests data from the Web server it originates from, and the Web server handles making the remote call to the third-party server and returning the results to the Web page. Although widely used, this technique isn't scalable.

Another way to overcome the SOP limitation is to insert a dynamic script element in the Web page, one whose source is pointing to the service URL in the other domain and gets the data in the script itself. It works because the same-origin policy doesn't prevent the insertion of dynamic script

elements into the Web page and treats the scripts as if they were loaded from the domain that provided the Web page. This is actually how JSONP works: load the JSON response within a <script> tag. In order to do this, the name of a callback function is specified as an input argument of the call itself. The remote server will then wrap the JSON response in a call to that function. When the browser finishes downloading the new contents of the <script> tag, the callback function executes. This approach requires the remote Web service to behave as a JSONP service; that is a Web service with the additional capability of accepting a callback function name as a request parameter and supporting the wrapping of the returned JSON data in a user-specified function call [1]. Popular service providers such as Google and Yahoo! already support this technique.

The simplicity of JSON as a lightweight data interchange format and the power of JSONP as a flexible technique for client-based (in-browser) mashups, make SMD much better suited to SMashups than SA-REST.

6 PLEF-Ext: SMashup PLEs with SMD

PLEF-Ext provides a flexible framework that can help learners easily extend their PLEs with new learning services. Driven by the SMashup concept, PLEF-Ext supports learners in sharing, finding, integrating, managing, reusing, and remixing semantically annotated RESTful learning services with minimum effort. PLEF-Ext uses SMD for the semantic description of the services. An abstract architecture of PLEF-Ext is provided in Figure 2. PLEF-Ext encompasses five main modules:

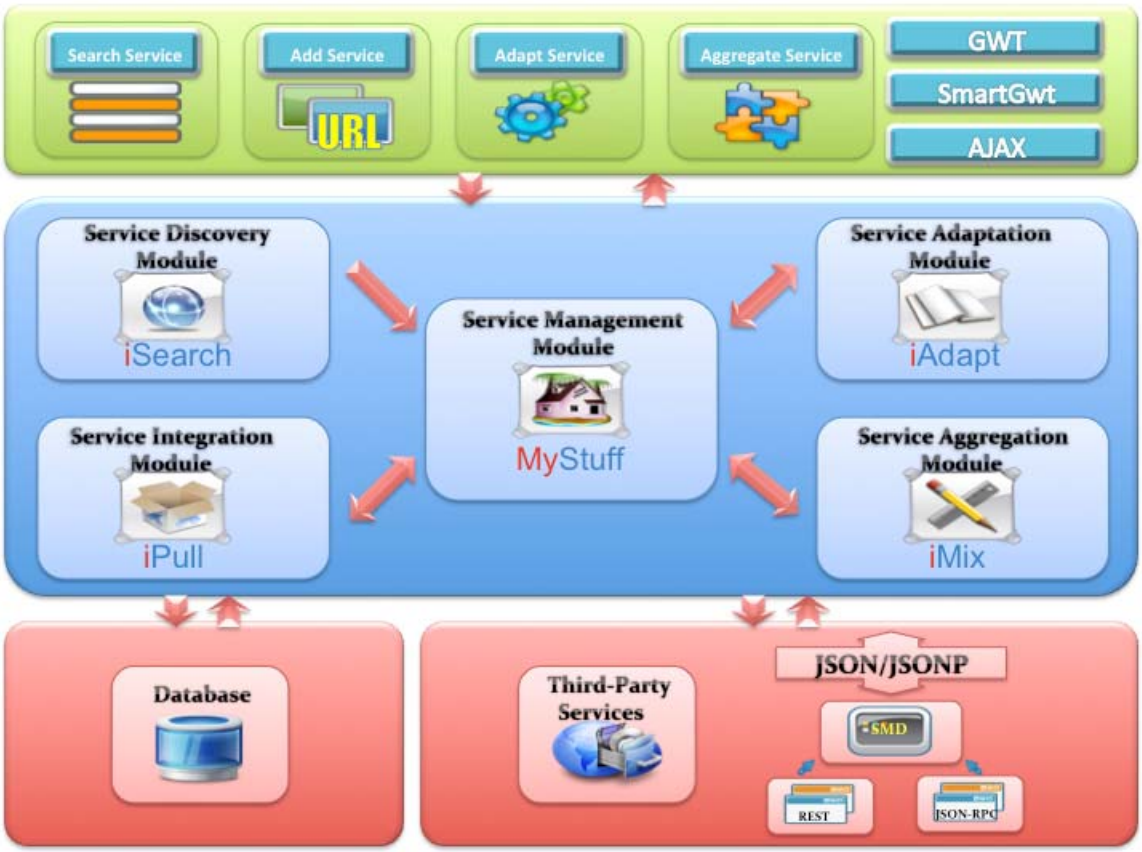

Fig. 2. PLEF-Ext Abstract Architecture

- The Service Discovery Module (**iSearch**) provides a tag-based search mechanism that enables learners to locate learning services remixed and shared by peers within PLEF-Ext.

- The Service Integration Module (**iPull**) makes it possible for the learner to easily plug in a new SMD-annotated external learning service into her PLE, by just specifying the URLs of the service and its SMD.
- The Service Adaptation Module (**iAdapt**) enables the learner to adapt a service to her needs, by just specifying the parameters and return values of the service.
- The Service Aggregation Module (**iMix**) lets learners compose a mashup query in a drag-and-drop fashion.
- The Service Management Module (**MyStuff**) supports the learner in managing (i.e. add, remove, edit, tag) her learning services within PLEF-Ext.

PLEF-Ext is built upon a client-server architecture. However, in contrast to traditional mashup environments, where the majority of computation-intensive processing will occur on the server (mashups are created using server-side proxies), the client in PLEF-Ext is much fatter. That is, the mashups will be built and run on the client-side, using the computing power of the client device. Coupled with JSONP services that expose their data in JSON and wrap it in function call, the client (browser) can make requests for data from the different services and use the fetched data to build in-browser mashups. In order to achieve this, the client in PLEF-Ext is assisted with an extensive user interface library of widgets and panels provided by SmartGWT[11] and powerful GWT[12] libraries to perform a wide range of actions, such as parsing of the fetched JSON data and asynchronous server communication through HTTP or remote procedure calls (RPCs).

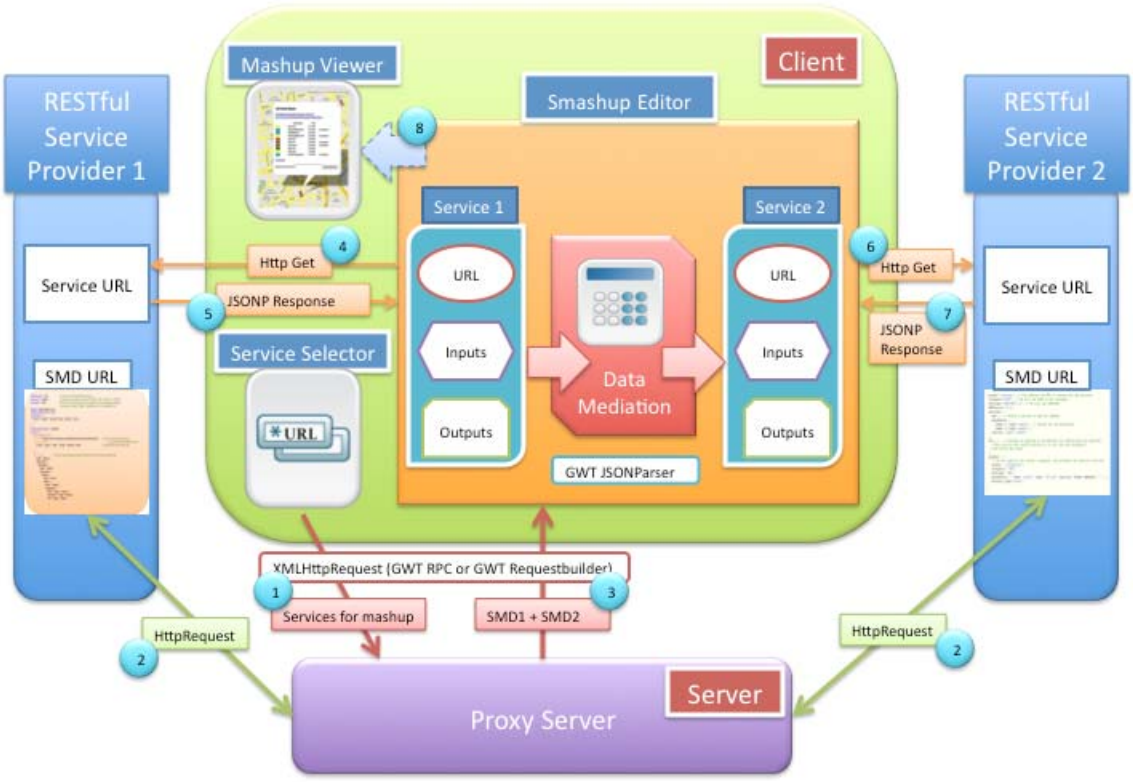

Fig. 3. SMashup Process in iMix

[11] http://code.google.com/p/smartgwt/

[12] http://code.google.com/webtoolkit/

In the following, we discuss in more details the functionalities of the Smashup module iMix. Figure 3 demonstrates the SMashup process in iMix. We suppose that the services to be mashed up are JSONP services. Their SMDs, however, are available on remote servers that return pure JSON data without wrapping it in a function call (since SMD is a relatively new concept, it is not supported yet by third-party service providers. For this work we are using SMDs that we created for popular JSONP services such as Google data APIs, Yahoo! search, geonames, delicious, and flickr). In this case, in order to work around the SOP security problem, we need to create a proxy on our local server. To note that if third-party service providers would in the future adopt SMD and would be able to return the SMD data via JSONP, the proxy server can be omitted and interaction with remote resources will be done from the client-side, solely via JSONP. This would then make the PLEF-Ext architecture much simpler. In order to mashup two RESTful Web services with iMix, a learner first needs to select the services to be mashed up. The client then makes HTTP calls (XMLHttpRequests) to the local server (proxy server) and have it go fetch the SMD data from the remote servers (step 1 in Figure 3). This can be achieved in GWT either with GWT RPC or direct HTTP using GWT RequestBuilder. The local server downloads the JSON-encoded SMDs from the remote servers (step 2) and passes it back to the client (step 3). When the client retrieves the SMDs from the local server, a JSON parser (GWT JSONParser) is applied to extract the descriptions of the services from their SMDs (i.e. service inputs, outputs, and request type) and display the descriptions to the learner in the SMashup Editor. The SMashup Editor is the key component of iMix. As mentioned earlier, a key diffculty in creating mashups is data mediation between the services to be mashed up. The main role of the SMashup Editor is to use semantic annotations to enable automatic data mediation without any programming knowledge from the learner side. The learner would only need to perform drag-and-drop actions using the Smashup Editor to define the smashup steps: (1) construct the query URL at which to invoke service 1, (2) construct the query URL at which to invoke service 2 and specify the mashup parameters (e.g. the outputs of service 1 that should be inputs for service 2), and (3) specify the outputs of service 2 (i.e. mashup result). Once the smashup steps have been defined, the learner can start the execution of the real mashup. At this point, the client invokes service 1 via JSONP to gather the data that are needed for the mashup (step 4 and 5) and invokes service 2 via JSONP to get the final result of the mashup (step 6 and 7), which will be shown to the learner in the mashup viewer (step 8).

7 Conclusions

In this paper, we discussed the challenge of PLE extensibility and how mashups can provide a powerful tool to achieve this challenge. We outlined the limitations in current mashup development platforms and tools, and discussed the benefits of Semantic Mashups (SMashups) as a flexible and scalable approach to creating mashups. We also discussed the Service Mapping Description (SMD) approach to adding semantic annotations to RESTful Web services. We finally presented the conceptual and technical details of PLEF-Ext as a mashup PLE framework that leverages SMD-based semantic annotations of RESTful Web services for user-friendly learning mashups. All of these efforts have been made available on the PLEF-Ext project homepage[13]. We welcome feedback from colleagues and users on both their experiences with the system and new ways they would suggest to leverage semantic mashup techniques to help learners build and extend their PLEs.

References

1. □ Ozses, S., Erg□ul, S.: Cross-domain communications with jsonp (2009)

2. Benslimane, D., Dustdar, S., Sheth, A.P.: Services mashups: The new generation of web applications. IEEE Internet Computing 12(5) (2008) 13-15

3. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, Chapter 5: Representational State Transfer (REST). PhD thesis, University of California, Irvine (2000)

4. Taivalsaari, A., Mikkonen, T.: Mashups and modularity: Towards secure and reusable web applications. In: 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshop Proceedings (ASE

[13] http://eiche.informatik.rwth-aachen.de:3333/PLEF-Ext/

Workshops 2008), 15-16 September 2008, L'Aquila, Italy, IEEE (2008) 25-33
5. Lathem, J., Gomadam, K., Sheth, A.P.: Sa-rest and (s)mashups : Adding semantics to restful services. In: Proceedings of the First IEEE International Conference on Semantic Computing (ICSC 2007), September 17-19, 2007, Irvine, California, USA. (2007) 469-476
6. Sheth, A.P., Gomadam, K., Lathem, J.: Sa-rest: Semantically interoperable and easier-to-use services and mashups. IEEE Internet Computing 11(6) (2007) 91-94
7. Taivalsaari, A.: Mashware: The future of web applications. Technical report, Sun Microsystems Laboratories (2009)
8. Sheth, A.P., Verma, K., Gomadam, K.: Semantics to energize the full services spectrum. Commun. ACM 49(7) (2006) 55-61
9. Verma, K., Sheth, A.P.: Semantically annotating a web service. IEEE Internet Computing 11(2) (2007) 83-85
10. R.Akkiraju, J.Farrell, J.Miller, M.Nagarajan, M.Schmidt, A.Sheth, Verma, K.: Web service semantics - wsdl-s (2005)
11. Zyp, K.: Service mapping description proposal (2008)