The Web Integration & Interoperability Layer (WIIL)
Turning Web Content into Learning Content using a Lightweight Integration and Interoperability Technique

Sokratis Karkalas, Manolis Mavrikis and Patricia Charlton
London Knowledge Lab, UCL Institute of Education, University of London, London WC1N 3QS, U.K.

Keywords: Learning Management Systems, Learning Platforms, Interoperability, Integration.

Abstract: This paper presents a technique that enables integration and interoperability of web components with learning platforms. This technique is proposed as a lightweight alternative to IMS LTI and OpenAjax and is especially suited to simple client-side widgets that have no back-end dependencies and potential security risks. The technique has already been used successfully in an experimental learning platform to provide data generated by various heterogeneous components for intelligent support and learning analytics.

1 INTRODUCTION

Educators have always been trying to take advantage of technology affordances of their time and introduce innovative approaches in their teaching. One major component of teaching and learning is the development and delivery of courseware material. Since TCP/IP and the advent of WWW a multitude of systems emerged as precursors of modern Learning Management Systems (LMS). Systems like TrainingPartner (by GeoMetrix) [1], Teachers Toolbox and Interactive Learning Network (by CourseInfo) [2] and ASAP (by ePath Learning) [3] attempted to leverage the potential of new technologies and offer efficient organisation, management and dissemination of teaching resources. The appearance of modern LMSs like Moodle [4], Blackboard [5], Sakai [6] and ANGEL Learning [7] and the development of standards like SCORM [8] changed radically the educational landscape (Bohl et al., 2002). In the 00s the level of acceptance and adoption started to dramatically increase and the LMS established itself as the dominant technology for more than a decade. Nowadays LMSs are mature and current implementations are stable, robust and reliable but that is just one side of the coin. LMSs ended up being treated like any other large-scale enterprise-wide application (Severance et al., 2010). The primary concern gradually shifted from education-related issues to considerations like system stability and reliability. As a consequence of that the process of integrating new functionality and instructional content became more difficult (Severance et al., 2010). The new challenge now is the ability to balance innovation with stability. The solution for stability was a shift to an architectural approach that offers the ability to decouple functionality into independent and self-sufficient components that interoperate via standardised communication protocols potentially over a network (González et al., 2009). Innovation is empowered by the ability and the freedom to combine potentially heterogeneous learning components into formations that offer new and unique learning experiences. The solution employed for the stability problem led to the development of a new market for learning components. These components are typically fully-fledged web-based applications equipped with their own infrastructure in terms of security and operations and able to provide their services as stand-alone applications. The need for these applications to integrate with LMSs without sacrificing stability led to the development of standards like the IMS Learning Tools Interoperability (LTI) specification and OpenAjax [9].

This is definitely a step forward but educators are

[1] http://www.trainingpartner.com/
[2] http://en.wikipedia.org/wiki/CourseInfo
[3] http://www.epathlearning.com/services/lms/
[4] https://moodle.org/
[5] http://uki.blackboard.com/sites/international/globalmaster/
[6] https://sakaiproject.org/
[7] http://www.angellearning.com/community/higher_ed.html
[8] http://www.adlnet.gov/scorm.html
[9] http://www.openajax.org/index.php

Karkalas, S., Mavrikis, M. and Charlton, P..
The Web Integration & Interoperability Layer (WIIL) - Turning Web Content into Learning Content using a Lightweight Integration and Interoperability Technique.
In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - Volume 2: KEOD*, pages 139-146
ISBN: 978-989-758-158-8
Copyright © 2015 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

still finding development in LMSs too restrictive for their purposes (Mott, 2010). LMSs are designed to be very controllable and well-structured. That makes them very efficient in supporting administrative functions but relatively inflexible in supporting student-centered learning scenarios. Nowadays, educators see learning platforms as highly customisable mashup applications that don't necessarily impose prefabricated and static teaching-centered material to students. Educators want the freedom to easily develop formations of components that are available on the web and make them part of their educational practice with little or no configuration overhead in a way that resembles systems like (Gurram et al., 2008). This is a browser-based application composition environment and run-time that simplifies development on top of existing complex systems. This new trend leads to systems that deviate from the basic LMS norm. These systems are called Personal Learning Environments (PLE) (Severance et al., 2008) or Personal Learning Networks (PLN) and are expected to be used in conjunction with LMSs. Another, more radical approach is the Open Learning Network (OLN) (Mott, 2010) that unifies both worlds in a single platform. The logic behind these systems is radically different and promises greater flexibility, portability, adaptability and openness but the stringent and expensive to implement processes of LTI still remain and have to be used even when practically they have nothing to offer.

In this paper we propose a new lightweight technique that can be used instead of LTI and OpenAjax for integration and interoperability of web components with learning platforms. This technique provides learning content authors the ability to utilise any type of web component with minimal development and administrative overhead. Furthermore, it promises robustness, better functionality and efficiency in every respect.

2 MOTIVATION

This work started as part of the MCSquared project (EU-funded). The purpose of this project is to design and develop an intelligent digital environment that enables authoring and utilisation of creative books (c-books). These are e-books that offer interactivity over rich media content and enhance creativity of mathematical thinking. The platform is deployed as a web-based application and features a flexible authoring environment that supports the dynamic integration of web components at design time. These components may be specialised learning widgets or any other type of component that provides a basic API and is available on the web.

A typical component of that category has the following characteristics:

1. It is deployed either as an individual widget or as a part of a JavaScript library that offers a logically interrelated collection of tools.
2. It is freely available and no copyright or license issues abide. Potential users are free to execute, copy, amend and distribute the software.
3. It offers an API through which its functionality can be made available to the users. Through this API it is possible to load, initialise, get/set its state and intercept user/system interactions with it.
4. It executes in the browser and there are no dependencies on back-end components.
5. It may include a visual part to be presented as part of the page.
6. There is no registration requirement for the component to be used.
7. It is hosted in a public Content Delivery Network (CDN) or it is downloadable and able to be hosted locally.
8. There is no ability to amend the implementation of the component. It is not possible or feasible to change its source and make it compatible with a potential host or extend it with some interoperability method remotely.

The motivation for this work was to devise a method that provides seamless integration of web components with the platform, with minimal technical support and administrative overhead. The method should be able to support efficient two-way communication with no server round trips (network traffic and back-end dependencies) and the implementation should be lightweight enough in order not to burden excessively the browser. The interface for this communication should be generic and able to support any type of standard like the W3C widget interface [10] or non-standard widget-specific interfaces. Cross-widget communication should be safe but not artificially constrained. It should be up to the implementer to decide what is exposed from widget interfaces and how it can be used by the rest of the system. In this work we are not concerned with general architectural issues regarding distribution of learning widgets for the web (Wilson et al., 2007; Wilson et al., 2008). Issues like widget packaging, deployment and description are beyond the scope of this project.

[10] http://www.w3.org/TR/widgets-apis/

The considerations that played a crucial role in this discussion can be summarised as follows:

1. `Component heterogeneity`: Nowadays the plethora of these components in the web is overwelming. There components are disparate and heterogeneous and their exposed APIs are always dissimilar. Integration with a platform requires a technique that is generic and independent of widget-specific functionality. The method should be able to overcome variability by providing a very simple interface, usable by any type of component.
2. `Platform compliance`: Another consideration is the potential to re-use the method in future platforms as well. The method should not be dependent on platform-specific functionalities and idiosyncrasies.
3. `Registration`: The integration process should not require the execution of protracted and cumbersome procedures. It should be possible to register the component with the platform with minimal effort and technical expertise.
4. `Communication`: Once the component is embedded in the platform, it should be relatively easy and inexpensive (in terms of resource utilisation and complexity) to exchange messages with its host. Passing messages should be based on a connectionless communication protocol and rely on system stability at both ends of the channel.
5. `Roles`: A crucial question is whether it is acceptable to consider the host (platform) and the guest (component) nodes as two equal entities in this relationship. In this case, implementation is simple and can be used globally by both sides in the same way. In the case that host and guest need to be treated unequal, the method should be based on the assumption that there are certain host and guest-specific functionalities that must be implemented. If this is deemed unnecessary, it obviously must be avoided.
6. `Browser security restrictions`: Modern browsers are not very tolerant with pages that intermix content from different domains and most web components will most likely originate from foreign domains. The method should be able to overcome security constraints and browser-specific idiosyncrasies.
7. `Performance`: Building a system as a dynamic and arbitrary collection of heterogeneous components, implies that these entities have their own space and distinct purpose in the system. It makes sense for these components to be able to operate in parallel and communicate asynchronously with the platform. In multi-processor systems this is not just a matter of asynchronous behaviour, but it can also make a huge difference in the overall performance of the application.
8. `Memory`: Memory footprint is becoming a serious issue in browser-based fat-client implementations. The interoperability part must not be a substantial burden in the memory balance.
9. `Security`: Security is always a major issue when integrating foreign and potentially non-trusted components with a system. The tendency is to create integration methods with artificial barriers in order to prevent developers from making dangerous mistakes. This approach obviously may have a major impact on the functionality that is eventually exposed and reused. The method in this project should allow for maximum flexibility in terms of what is exposed and what is not. It should allow both secure containment of unsafe material and unrestricted exposure of data and operations wherever needed. It should be up to the designer/developer to decide what is secure and what is not.

3 THE METHOD

The method devised can logically be thought of as a combination of two parts: communication protocol and node interfacing. The terms we use to identify the interoperable parts (nodes) are host and guest. The nodes are treated as equals and two-way unrestricted communication between them is assumed. The implementation is based on a thin JavaScript wrapper that abstracts the node implementation from its interface and encapsulates its internal specifics. The wrappers provide the ability to selectively externalise any part of the nodes' functionality in a generic way. The functionality is exposed through the definition of a public interface that maps internal implementations to publicly available methods. These methods can then be callable by the communicating parties through message passing.

3.1 Browser Security

A major challenge when integrating content from different domains over the web is the `Same-Origin Policy` (SOP) enforced by all modern browsers. This is a policy that aims to prevent unauthorised access to confidential information by malicious scripts and thus protect data integrity.

A simple solution to the problem is to reference the components directly as JavaScript libraries in the

host page. The origin in this case is defined by the location of the containing page. Therefore, even if we have to load multiple components from various origins, the files will eventually run in the origin of the page that includes them. The biggest problem with this scenario is the possible use of mixed content in the case of files coming from both secured (HTTPS) and non-secured (HTTP) origins. Behaviour in this case is browser-dependent and typically problematic. Another problem is that code, regardless of origin, will run under the same context as a single-threaded application. Performance-wise this is not desirable. The third issue is code organisation. Intermixing code from different sources in the same global namespace is a potential risk. Accidental name clashes that invalidate data are not uncommon problems in this case. In conclusion this method is obviously not an option.

Another method for performing cross-origin requests is `JavaScript Object Notation with Padding` (JSONP). This is essentially a hack based on the premise that JavaScript code referenced directly from a page, eventually runs in the origin of that page. This method presupposes a great deal of control over the component source and JSONP-aware services. It also suffers from most of the problems mentioned above. For these reasons this approach is inadequate for our purposes. A third method (that is also a hack) is to circumvent the policy by not making any cross-origin requests at all. This apparently requires an extra logical tier in the system that resides at the server side. Requests are sent to a server-side proxy that has the same origin as the page. This approach is cleaner than the previous ones but requires an extra server-side component which makes it somewhat cumbersome. Another alternative is to use `Cross-origin Resource Sharing` (CORS). The assumption again is that we have access over the server processes and we can overcome SOM by adding an HTTP header in the response. This header can then instruct the browser not to consider the call a SOM violation. Apart from the previously mentioned issues, this approach has the additional problems of potential browser incompatibility and header removal by firewalls. Modern browsers have the ability to bypass SOP by making calls to WebSocket addresses. In this case it is the WebSocket server that does the security checks and allows the caller to receive an answer or not. Server dependencies and other obvious problems make this approach equally inadequate as the above. A fifth method is to use iframes by explicitly declaring in JavaScript that the nodes have the same origin. This can be done by setting the property `document.domain` to the same domain name at both ends. A system called Subspace (Jackson and Wang, 2007) is using this technique to implement cross-domain communication for web mashup applications. The actual behaviour depends on the browser and another problem is that resetting the domain property may not have the expected result. A typical problem is that the port number may be set to `null` (empty) by that process. Implementation is again browser-dependent.

The only method that overcomes all of the above obstacles is to embed the component in a sandbox and communicate through HTML5 messaging. In HTML terms this can be a common `iframe` element that hosts a separate page containing the external library. The component is kept isolated in the sandbox and executes in its own context as a separate application. That provides the advantages of code safety and parallel execution. The component interoperates with its host through a messaging system inherently supported by HTML5 (Järvinen, 2011). Concurrency is maintained by asynchronous message passing at either direction. Execution and data interchange take place entirely in the browser and there is no network and server overhead involved.

3.2 Interfacing

As explained above, node diversity is hidden within a wrapper. The wrapper provides a very generic interface through which basic communication can be carried out. The interface comprises the following two functions:

- sendMessage(message)
- receiveMessage(event)

This system provides the ability for two-way communication between the host and the guest. In both cases the data is sent in the form of a `message` object. The only difference is that in the latter case the `message` is received as a property of an `event` object. Message passing in the HTML5 system is carried out using events.

The format of the `message` object follows:

- `origin`: This property serves as the unique identifier of the component that sends the message. It is not the same as the homonymous property of the event object that carries it when in transit. The latter corresponds to the domain of the sender. This property is just some text that uniquely identifies the component in the system.
- `content`: This property can contain data of any type. The purpose in this case is to send some data and let the receiver decide what to do with it.

- `command`: This is an instruction (command) that is possibly sent along with some data in the form of arguments. The intention in this case is to utilise receiver-specific functionality and perform some processing there. This functionality is exposed in the form of a public interface that the receiver makes available to other communicating parties. The command can only be given as a `string` (text).
- `args`: This is an array of values that accompany the command. These values are addressed to a method of the receiver's public interface. That method is what the command property refers to.
- `callback`: This is a `string` value (text) that corresponds to a function exposed in the public interface of the sender. The receiver, upon receipt of the message, performs the requested operation and then sends the resulting value as argument to the callback function of the sender. This property permits the asynchronous continuation of the same logical process in the sender after a remote call in the receiver is completed.

This message gives the ability to either pass some data to another node for internal consumption or instruct the other node to execute a function (remotely) and possibly send back the results. Since the communicating parties do not have direct access to each other's APIs, the instruction (command) can only be sent in the form of text. The receiving wrapper uses this text to identify the actual function that needs to be called and request the operation.

A message object is not valid if:

- `command` is not a string
- `args` is not an array
- `callback` is not a string
- both `content` and `command` are not given
- `args` is given without a `command`
- `callback` is given without a `command`

The above validation rules are enforced by wrappers. Components by their nature are expected to be very diverse in terms of functionality. As a consequence of that their interfaces are expected to be heterogeneous. On the other hand platform - widget interoperability should be based on a standardised uniform way of communication. The solution to this problem is to enable interprocess communication through the above `message` objects. Instead of extending the basic interface presented above with more functions, we pass method names and parameters as properties of messages. By using this technique we keep a simple and uniform interface that facilitates basic communication for any type of node (component or platform) and at the same time we accommodate the utilisation of the nodes' particular functionalities without constraints.

According to the above system every wrapper must define an object called `publicIF` that exposes the node's particular functionality to the rest of the world. This functionality can be utilised locally by calling this object's public member methods. The same methods can also be called remotely through a special executor method that is also provided by `publicIF`. If, for example, there is a method called add(a, b) that performs addition, this method can be called in two ways:

`locally`: publicIF.add(a, b)

`remotely`: publicIF.execute({'command':'add', 'args':[a, b], 'callback':'log'})

The only argument of the executor is the message object described above. The information about the actual function to be executed, its arguments and the callback is given as properties of the object.

3.3 Communication Protocol

The communication protocol for platform - widget communication doesn't have to be particularly complex. There are four communication scenarios that can possibly take place:

a. The guest wants to inform the host about its availability. This message is supposed to be sent immediately after the guest loads up and is fully functional within its page. The message follows:

`content`: 'ready'
`command`: null
`args`: null
`callback`: null

b. The node (guest or host) wants to send a message to the other party without any instruction as to what the receiver should do with it. This is a simple message with some content (like the previous one).

`content`: 'some content'
`command`: null
`args`: null
`callback`: null

c. The node (guest or host) wants to use a service provided by the other party. That entails the execution of a remote method. An answer may be required as well. The message looks like the following in this case:

`content`: null
`command`: 'add'

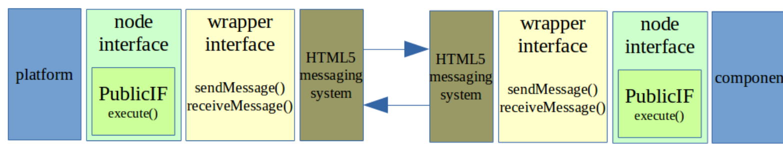

Figure 1: The interfacing stack.

`args`: [2,3,4]

`callback`: 'display'

The receiver is expected to perform the operation and return the result to the caller (enclosed within another message). The new message takes the form of another function call to the caller's `display` function.

d. The node (guest or host) wants to send some data and instruct the receiver explicitly what to do with the data. That, again, entails the execution of a remote method. The data is send as an argument for the remote method to be executed. The message would look like the following in this case:

`content`: null

`command`: 'logActions'

`args`: [{action1},{action1},..,{actionN}]

`callback`: null

It is, of course, up to the implementer of the integration to decide what the protocol should be able to do. In the scenario presented above the assumption is that once the guest becomes fully functional, an uninterruptible (HTML5) communication channel becomes available. If the host knows that the guest exists and is available, then it is safe to assume that the guest will remain available throughout the whole session. But that may not be true if the guest crashes or the element holding the sandbox is removed from the host's DOM for some reason. The communication protocol could be used in a less connection-less manner in this case and check for availability at certain time intervals.

3.4 Component Installation

Third-party components are typically considered external to an application and therefore an installation is required prior to their use. This type of process can take many forms in web-based applications. In Moodle, for example, plug-ins must physically become part of the application codebase. If the component is hosted externally and is LTI-compliant, there is a registration process that provides configuration parameters and a method of authentication (OAuth). Configuration parameters typically include the URL referencing the tool, the user credentials under which a trust relationship can be established (consumer key and shared secret in Moodle) and launch instructions.

Our system is designed to work with external non-LTI-compliant components. Installation is much simpler and registration is more lax since authentication is not required. A mutual trust relationship can be established by injecting the trusted foreign domain names to the nodes.

3.5 Component Launch

The LTI Launch protocol can be a long-winded process for non-LTI-compliant components. Making a simple client-based web component LTI-compliant just for the sake of making it interoperable with the platform is an overkill. The LTI launch process is depicted in figure 2.

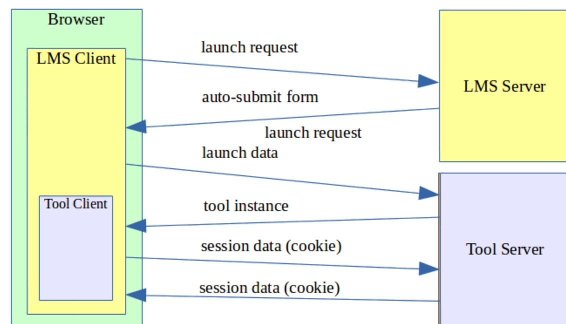

Figure 2: The LTI launch protocol.

The tool is selected by the user in the LMS environment (browser). The LMS server prepares the necessary information for the launch as a HTML form and sends it back to the browser. Upon arrival the form gets automatically submitted to the tool. The user gets authenticated and the tool provider sends back a tool instance. After that, session information is maintained in cookies during server roundtrips.

If the component does not include any native server-side logic, then according to the LTI launch protocol, some server-side code must be introduced. The self-submitting form that contains launch information is sent to the server through a POST HTTP method. There has to be something at the back-end to receive and process the values. Apart from the extra processing tier, this method entails unnecessary network traffic.

In our system, the launch protocol is much simpler. The tool is invoked by the user in the LMS en-

vironment and a request is sent to the tool provider. An instance of the tool is returned and loaded into the guest's DOM. The guest informs the host that the tool is ready to be used and the host sends a message with initialisation data. After that the tool sends data updates whenever user activity is detected. The process does not involve any server-side processing and network traffic is not incurred. After instantiation the tool communicates with its host locally through message passing.

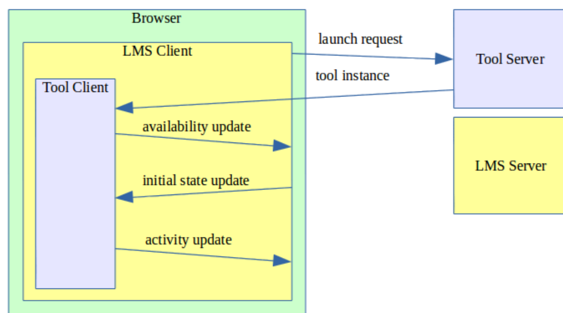

Figure 3: A simpler launch protocol.

3.6 Cross-component Communication

Typically the framework used for this part does not deviate a lot from the basic principles of OpenAjax. In the OpenAjax world cross-component communication takes place through managed or unmanaged hubs. A component may take the role of a producer that publishes messages to the hub and/or a consumer that subscribes to receive messages from the hub. The hub is designed around the concept of anonymous broadcasting. Producers and consumers are not aware of each other. Point-to-point messaging, cross-component property management and remote procedure calls are not inherently supported.

In the approach proposed here communication between components is possible only through the platform's wrapper. In that respect this wrapper plays the role of a hub. Components live in their own secure environment (sandbox) and exchange information with the platform through message passing. This is where the similarities end.

An important difference is that the platform itself is a component. In this case communication is direct and unrestricted. System integrators are allowed to expose a widget functionality (or part of it) and make it available to its host and vice versa. Components are able to exchange messages and to make remote procedure calls. Property management is also possible through the same mechanism. Communicating parties are fully aware of each other's exposed functionality and are free to utilise it. A distinguishing feature of this system is the ability to asynchronously execute a logical process in the sender after a remote procedure call is completed.

4 A WORKED EXAMPLE

In this section we present a sample application that demonstrates a basic but complete integration scenario. The guest in this case is a page that hosts a learning activity developed in Geogebra. The host is just a simple page that coordinates the operations. The wrapper of Geogebra in the guest registers a few event handlers and intercepts user interactions with the tool. In this activity the student uses the sliders to change values in variables. User activity data along with the current state of the construction are sent through the wrappers to the host.

```
updateObjectHandler = function(object)
{
var args={'object':object,'action':'updated'};
var message =
createMessage(null,'logAction',[args],null);
sendMessage(message);
}
```

The only thing that needs to be done in the host is to implement and expose a function that processes the input data.

```
//private method
function logAction(action)
{
  database.log(action);
}
//public method
publicIF.logAction = logAction;
```

The actual data is received in the form of an event object but the wrapper transparently handles unpackaging of the content and delivers the data directly to the requested method.
The host inserts the data into a local JavaScript database that is linked to visualisations on the page. As the student interacts with the tool and the data changes in the database, the host displays real-time user activity/performance statistics in histograms. The visualisations are themselves separate widgets hosted in their own guest sandboxes and communicate with the host using the same system.
One additional feature of this implementation is that the host analyses the data dynamically using a rule-based expert system and provides real-time intelligent support to the student. If the student achieves something that seems to be leading to the correct direction, the host displays a message to reinforce this attempt.

Figure 4: The 'Ladders' Activity from Geogebra Tube.

The student can also ask for help and check whether the objective has been accomplished or not. All this processing uses data coming dynamically through the interoperability sub-system.

5 CONCLUSION

The technique presented in this paper has been used in an experimental system that was developed as part of the EU project MC2 [11] at London Knowledge Lab, IOE UCL [12]. Preliminary test results showed that the method works as expected and fulfils the original design goals. The method deals effectively with component heterogeneity and seamlessly integrates disparate components into a seemingly homogeneous whole. Registration, instantiation and initialisation of these components is simple and efficient. Operation is safe and the system performs well when the components asynchronously communicate with the platform. The method overcomes browser security restrictions and the overhead in terms of memory and processing power needed is minimal. It is estimated that the experimental implementation has successfully processed so far approximately 37,000 events. Sample tests showed that messages are being exchanged with 0% loss at a speed that allows a very smooth interaction between different components. In future versions of the system we envisage to implement an on-line editor that simplifies the integration process by inserting wrapper boilerplate code to the nodes and by providing the ability to visually manipulate them.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement N°610467 - project "M C Squared". This publication reflects only the author's views and the EU is not liable for any use that may be made of the information contained therein.

[11]http://www.mc2-project.eu

[12]http://www.lkl.ac.uk

REFERENCES

Bohl, O., Scheuhase, J., Sengler, R., and Winand, U. (2002). The sharable content object reference model (scorm)-a critical review. In *Computers in education, 2002. proceedings. international conference on*, pages 950–951. IEEE.

González, M. A. C., Penalvo, F. J. G., Guerrero, M. J. C., and Forment, M. A. (2009). Adapting lms architecture to the soa: an architectural approach. In *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on*, pages 322–327. IEEE.

Gurram, R., Mo, B., and Gueldemeister, R. (2008). A web based mashup platform for enterprise 2.0. In *Web Information Systems Engineering–WISE 2008 Workshops*, pages 144–151. Springer.

Jackson, C. and Wang, H. J. (2007). Subspace: secure cross-domain communication for web mashups. In *Proceedings of the 16th international conference on World Wide Web*, pages 611–620. ACM.

Järvinen, H. (2011). Html5 web workers. In *T-111.5502 Seminar on Media Technology BP, Final Report*, page 27.

Mott, J. (2010). Envisioning the post-lms era: The open learning network. *Educause Quarterly*, 33(1):1–9.

Severance, C., Hanss, T., and Hardin, J. (2010). Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools. *Technology, Instruction, Cognition and Learning*, 7(3-4):245–262.

Severance, C., Hardin, J., and Whyte, A. (2008). The coming functionality mash-up in personal learning environments. *Interactive Learning Environments*, 16(1):47–62.

Wilson, S., Sharples, P., and Griffiths, D. (2007). Extending ims learning design services using widgets: Initial findings and proposed architecture.

Wilson, S., Sharples, P., and Griffiths, D. (2008). Distributing education services to personal and institutional systems using widgets. In *Proc. Mash-Up Personal Learning Environments-1st Workshop MUPPLE*, volume 8, pages 25–33.