

Learning Design, generic service descriptions and universal acid

Martin Weller, Alex Little, Patrick McAndrew and Will Woods

The Institute of Educational Technology, The Open University
Milton Keynes MK7 6AA, United Kingdom
m.j.weller@open.ac.uk

ABSTRACT

This paper examines the contention that learning environments which use IMS Learning Designs can be created by plugging in different components, using generic service descriptions to create the interface between the Learning Design (LD) and the specific tools. There is an alternative viewpoint which claims that generic service descriptions cannot provide the richness required to fully utilize Learning Design. The paper describes the work performed in the SLeD project by the UK Open University and the Open University of the Netherlands. The SLeD project suggests a compromise between the two viewpoints by using generic service descriptions, but recognizing the nature of the current environment through the use of translators, which interact with specific instantiations of services.

Keywords

Learning Design, service descriptions

Introduction

Over recent years there has been a considerable push towards interoperability, both within the educational sector and in terms of broader data exchange. This desire for interoperability has several motivations underpinning it. Perhaps primary amongst these are cost considerations. As it became evident that elearning was not a cheap alternative to face to face teaching, then the desire to reuse grew (Weller, 2004). The initial focus of reuse was on content, with the notion of learning objects, and building an 'educational object economy'. While interest in learning objects and repositories continues, they have not seen the large-scale uptake that many predicted. Barriers to the success of content reuse were identified as the granularity of the objects, the ownership attached to the different levels of the original design and possible reinterpretations (Laurillard & McAndrew, 2003). The Learning Design specification (IMS, 2003) seeks to provide a means of reusing the pedagogy, or design, of a learning activity, and not just its content. Koper & Olivier (2004) suggest that Learning Design (LD) can be used as a means of representing and encoding learning materials, and this is especially suited to the elearning context while neutral to the pedagogy that is being applied. This may go some way to addressing some of the barriers to uptake found in a purely content-centric view of reuse, embodied in the learning objects paradigm.

As well as reusing content and design, it makes financial sense to reuse software components in the development of larger systems. A related motivation is the convenience afforded by reusing existing components that have already been developed and tested, instead of creating each one from scratch. The rise and acceptance of open source software developments has also suggested a third motivation, namely that of quality. By allowing components to be reused in different contexts they are improved or adapted by a community of users, to become increasingly robust.

Within the educational specifications area the initial focus was also on the reuse of content, with the resulting standards providing means of describing resources (metadata) and structures of resources (content packaging). More recently the focus has shifted towards interoperability of tools and services, as evidenced by the initiation of a Tools Interoperability Specification by IMS. However, beyond the generic web services standards, this is still an immature area with few robust and reliable standards that can be used.

The attention on tool interoperability has potential impact in the Virtual Learning Environment (VLE) sector, as educators begin to consider component VLEs comprised of a number of best of breed components, instead of the more integrated, monolithic systems offered commercially. The viability of such component VLEs has been raised by recent developments which seek to specify a generic, standards-based approach to VLEs, often focused around open-source systems. These include the SAKAI initiative in the US and the JISC service oriented architecture in the UK. The SAKAI project (<http://www.sakaiproject.org>), aims to deliver the following all as open source:

"The products of this project will include an Enterprise Services-based Portal, a complete Course Management System with sophisticated assessment tools, a Research Support Collaboration System, a Workflow Engine, and a Technology Portability Profile as a clear standard for writing future tools that can extend this core set of educational applications."

The JISC framework (Wilson et al., 2004) outlines the benefits and approach for adopting a service oriented architecture, which can be seen as a means of viewing the integration of systems:

“When we embark on this kind of analysis, identifying the parts of the MLE at a more granular level than monolithic systems, then we eventually end up with a *framework* of service descriptions. We are no longer interested so much in replicating data between large systems, but instead focus on what kinds of services are needed in the overall architecture to provide certain kinds of behaviour from applications.”

Such an approach to services and tools is especially relevant to a Learning Design perspective. If the reuse of learning designs is to be realized, then it is likely to be because they meet the three main motivations for reuse set out earlier. They provide savings, can be more convenient than creating from scratch and offer quality benefits. Such learning designs are likely to be reasonably complex and pedagogically rich, since relatively simple ones can be easily created, thus reducing the benefits of reuse. This complexity of structure will often lead to a requirement for the use of a range of tools and services. Currently only email, conference and search are specified in the Learning Design guidelines. In order for complex designs to be created a greater range of services needs to be described, along with the provision for adding to these.

If learning designs are to be reusable however, they need to remain neutral in terms of requiring specific tools. The service approach therefore holds great attraction for the Learning Design community, as environments configured in this way have a greater potential to accommodate a Learning Design approach by calling on specific instances of services.

In order to realize this, three factors need to be in place:

1. Generic descriptions of services that a learning design can interpret in order to create complex pathways through material. For example, all bulletin boards perform the same sorts of functions. By describing these, a design that utilizes a bulletin board for an online debate with different roles (e.g. proponent, opposer, scribe etc.) can be realized.
2. A methodology for describing these services so that new ones can be added. This needs to encompass the means by which services are described, how LD recognizes these and how the description or consensus about a description is arrived at.
3. Tools, services and environments that are amenable to such an approach. This will include being able to expose the main functions of a tool, for example through open APIs or web services.

However, creating a generic service driven architecture is not easy. Despite much of the discussion surrounding this approach, there are few successful implementations. The Tasmanian LEAP project is a rare example (LeAP, 2004) which uses a service oriented approach to create a flexible VLE:

“The project has guiding principles of interoperability and the use of standards for data and infrastructure. The preferred application architecture model uses a “service based infrastructure” approach. The reality is that the diversity of products within the educational computing environment makes it impossible to adopt a single approach to application architecture. LeAP considers it good practice to use existing services and create new services as application development progresses.”

This may simply be a reflection of the relative immaturity of this approach. SAKAI is a relatively new consortium and have given themselves a tight timescale to deliver the first version of their vision. However, the lack of large-scale robust systems deploying generic service descriptions may also point to more fundamental problems, and maybe it remains an attractive theoretical construct that is difficult to realize in practical terms.

The question facing those working in this field then is to what extent a generic service description approach is achievable, and practical? If it is realizable then there are important implications for elearning, since it allows best of breed environments to be developed. As Wilbert Kraan of CETIS (2004) comments:

“It is becoming clear that common e-learning activities ... can't really be done by one application that has little or no knowledge of everything else on the network or the wider internet. It's also becoming clearer that a single system that tries to combine all such functions is unlikely to do all of them equally well. Furthermore, one size systems do not necessarily fit all institutions.”

However, the success of a generic services approach has wider software applications also. A generic service approach has the ability to influence what happens across a range of domains. In his book *Darwin's Dangerous Idea* Dennet (1995) proposes the notion of a universal acid which is so strong no container can hold it. He uses this analogy for the theory of evolution, demonstrating how it was not refined to biology alone. In a more limited sense, a generic service approach has this ability also, since it demonstrates that rich environments can be created

from components that are not tightly integrated, and can be decoupled easily. The analogy of Dennet's universal acid is also applicable in that he argued that evolution demonstrated how complex and rich variation in living species could be derived from relatively simple processes, in essence that relatively simple algorithms can produce complex behaviour. The generic services approach similarly claims that complex and rich environments can be developed from simple service descriptions, without the need for programming complexity.

The opposing view is that while generic service descriptions are appealing from a theoretical and architectural perspective, they are impractical and inefficient. For example, in developing the LAMS tool, Dalziel (2005) suggests that in order to create tools that are meaningful from a Learning Design perspective – 'Learning Design aware' tools as he terms them – it was more practical to build the tools from scratch than reengineer existing ones. LAMS provides the educational author with a number of tools, such as voting, discussion, quizzes, etc. Each of these components was built specifically for the LAMS editor, so that the sequencing of activities that LAMS sets out can be realized. Dalziel makes the distinction between 'rich' and 'minimal' component integration, arguing that for the necessary control and flow through a Learning Design driven environment, rich integration is the better option:

“Richly integrated components, as demonstrated in LAMS, are technically more challenging to achieve initially, but provides a seamless, integrated environment for both teachers and learners, with better potential for reliable quality of service.”

A generic description will always offer less functionality than a complete, bespoke service description and so much of the richness of individual programs is lost. In addition the brokering of services required in such an approach leads to inefficient data handling and needless additional steps in the transaction path.

The SLeD project

The SLeD project (<http://sled.open.ac.uk>) aimed to address some of these issues. The project was funded as part of the JISC Elearning Framework programme (<http://www.elframework.org/>), which is itself constructed around the concept of a service oriented architecture. The project was a collaboration between the UK Open University (UKOU) and the Open University of the Netherlands (OUNL).

The specific objectives of the project are given below, but the more general purpose was to extend the tools available to the Learning Design community and also to further develop our own understanding of how a Learning Design approach might be practically realized within an institution. For both the UKOU and the OUNL, Learning Design has three possible benefits:

1. As a means of describing course design in a format that can be shared between academics and technical staff.
2. As an audit trail of the design decisions, which can be reviewed as part of any quality assurance process.
3. As a means of providing structure and support to students and tutors via the delivery mechanism. This is particularly acute when students are studying at a distance and attempting complicated activities.

The SLeD project can thus be seen as an attempt to address at least part of this growing institutional interest in Learning Design (McAndrew & Weller, 2005).

The initial project was focused on upgrading the OUNL CopperCore Learning Design engine (<http://www.coppercore.org/>), to deal with level B learning designs, become SOAP compliant that it could utilize a web services approach and to develop a Learning Design player. The specific objectives were:

1. Upgrade CopperCore Learning Design engine to be SOAP compliant.
2. Upgrade CopperCore to be IMS LD level B/C compliant.
3. Produce a service based player system (SLeD) linked to the CopperCore engine.
4. Create ancillary services for the player to control rendering and environment information.
5. Enrich understanding of learning design in the context of course production and evaluate Learning Design as a method of supporting online course development.
6. Create a Wizard and guidelines for authoring.

A Learning Design player was already included in the CopperCore package, but the rationale for this project was to separate out the player functionality from the underlying engine. This was partly realized within the existing CopperCore Engine but the focus of the JISC-funded project was to enhance the ways in which the CopperCore Engine could be used by adopting the web services approach. So, communication between the player and engine used web services and additional end user tools such as search and conference systems were coded to allow web

services. This approach proved useful in extending the reach of the CopperCore system by allowing others to take advantage of the changes to CopperCore, for example the Reload system (<http://www.reload.ac.uk>) is now able to prepare a 'dummy run' which can be validated and tested using the same communication routes with CopperCore.

This project coincided with the OUNL roadmap for enhancing CopperCore capabilities to support the higher levels of LD. Level A support means that a complete planned design can be presented provided its structure can be pre-determined for each role within it. This allows for fairly linear learning designs, but not more complex ones. Supporting level B means that designs can include properties and conditions that determine progress in a more dynamic way. These changes led to the release of an updated reference player to demonstrate the new features and release of a separately developed player SLeD to demonstrate new ways to communicate with the engine and provide a path for less constrained development of the player system. By separating out the player and the underlying engine it maintains a 'clean' architecture that embodies the modular principles of the initial CopperCore project.

Initial work was successful in integrating two types of conference (forums) into the SLeD player, and similarly two separate instantiations of a search function. The two forums were OpenText's FirstClass system, and the inhouse Knowledge Network at the OUUK. The two search tools were Google, and the Knowledge Network again. The integration of these services demonstrated that both commercial and open systems could be called from the player, giving users a wide choice as to the actual implementation of any service they prefer.

One objective that was not met was the development of a wizard-based approach to developing LDs, although some initial work was begun in this area. This is currently being developed in a separate JISC funded demonstrator project.

A second project was initiated in April 2005, with further funding from JISC. The aim was to build on the success of the first project, particularly in extending and formalizing the approach for integrating services while maintaining the architectural integrity of the system, and also to continue the development of the CopperCore engine. There were three main deliverables for this project:

- Upgrading of CopperCore to integrate QTI calls in LD packages
- Development of a technical methodology for integrating service calls in LD
- Demonstration of this methodology with an interface to an ePortfolio tool

The project sought to significantly extend the initial SLeD work in two key areas. Firstly, although Learning Design had generated a lot of interest and enthusiasm, it was now at the stage where it needed to be put into practical use. The integration of assessment services into the learning designs was seen as a crucial factor in this. By upgrading CopperCore to validate IMS LD packages containing QTI this significant advance could be realized through a recognized core Learning Design system.

The second key area was to address the area of current weakness in the Learning Design specification, namely the paucity of services which it can reference. The work in the SLeD project began to address this by developing a generic method for calling search and conferencing services. The second phase of the project further developed this approach and formalized it, to create a toolkit for service integration. The proposed approach was to use the QTI work as a test bed to develop a generic technical methodology for integrating services.

Through this project it was hoped that an answer, or at least further insight, would be gained to the question as to which of the two views outlined above regarding generic service descriptions was correct. Is it the universal acid or the impractical concept?

SLeD Architecture

The experience of implementing bulletin boards in the initial SLeD project, and the work in integrating QTI calls led the project team to devise the architecture shown in figure 1.

The shading in the diagram represents the work of the second SLeD project, in extending the work undertaken in the initial project.

In this model the generic service descriptions are housed in the services broker. A learning design can also be delivered with other valid packages, for example QTI files, inside a single content package. The IMS LD

Content package contains and defines all data for all the required services. The Learning Design engine (in our case CopperCore) is responsible for the validation of the IMS CP package and the correct publication of it to the different services. The service dispatcher interprets the type of resource requested by the player and acts upon it. It contains the logic for synchronizing the properties and calling the underlying services. In the case of the SLeD project the Learning Design Engine will be CopperCore, and the player is SLeD, but in this open architecture these could both be replaced. Similarly, the QTI engine should be any standards compliant engine. Other services might include forums (or bulletin boards), eportfolios, search, email, etc.

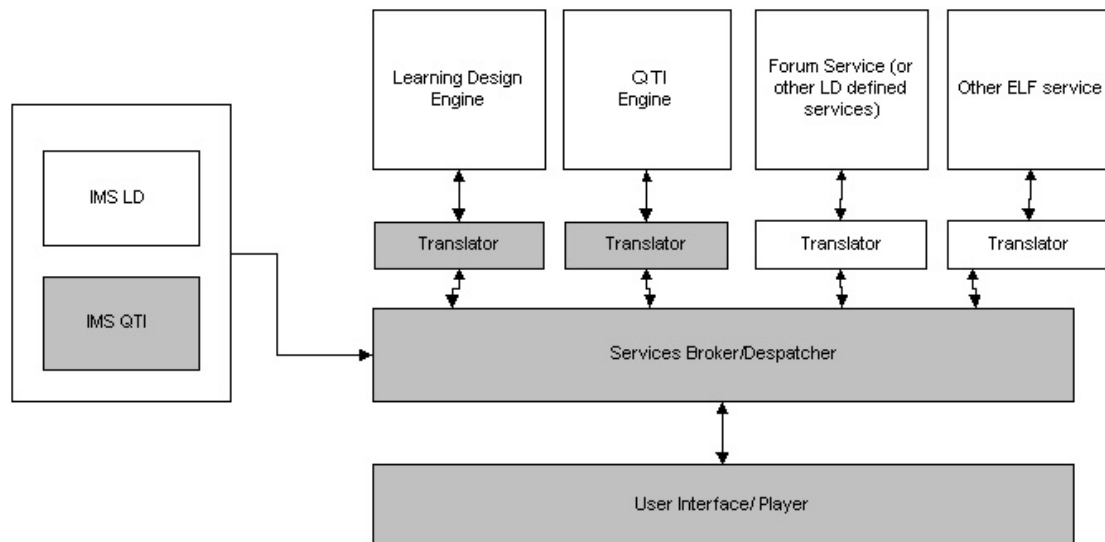


Figure 1. Structure of SLeD

The player handles the display, coordination and user interface of the services to the user.

Although generic service descriptions are stored in the broker, unless each individual service is set up to pass information back in exactly the required format, the generic descriptions will not be able to correctly handle the data. It is therefore necessary to have a small piece of application specific code that in effect acts as a translator between the application in question, and the generic service description.

Essentially the translators provide the interface between the broker and the actual service. The purpose of the translator is to match up the method calls made by the broker to the methods provided by the service provider, which may be internal java, web services, etc. A simple example would be that the APIS QTI service (<http://ford.ces.strath.ac.uk/APIS/>) might have a method 'getQuestion', but another QTI service might have called a method (which serves the same purpose) 'returnQuestion'.

Each of the translators is an individual java library (.jar file), not just a separate class within the application, as having separate files means that if someone wishes to use a different application for an existing service, then they simply replace the translator file.

Another advantage of having separate java libraries is that different installations can use different combinations of services, but still maintain portability of the translators. If everything (broker & connections to services) is bound up into a single library, it would require more work to switch between service providers and would greatly reduce the portability of the components.

The amount of coding required for a translator is relatively small, an example for a search translator which uses the UKOU's Knowledge Network (an internal document management system) is given below:

```
public String doSearch(String SearchTerm){
    String ret = "";
    try {
        // KN Search
        String endpoint = searchConfig.getSearchWebServiceURL();
        Service service = new Service();
        Call call = (Call) service.createCall();
        call.setTargetEndpointAddress(new java.net.URL(endpoint));
        call.setOperationName(new QName("http://webservice", "KNSearch"));
    }
}
```

```

        Object[] inParams = new Object[1];
        inParams[0] = new String(SearchTerm);
        java.lang.Object response = call.invoke(inParams);
        ret = formatKNResults(response.toString());
        service = null;
    }
    catch (Exception e) {
        ret = "Error in doSearch: " + e.toString();
    }
    return ret;
}

private String formatKNResults(String in){
    String output = xmlDeclaration + "<search>";
    try {
        SAXBuilder builder = new SAXBuilder();
        Document inXML = builder.build(new StringReader(in));
        List runList = inXML.getRootElement().getChildren("result");
        for (int i = 0; i < runList.size(); i++) {
            Element temp = (Element) runList.get(i);
            output = output + "<result>";
            output = output + "<rank>" + temp.getChild("rank").getText() + "</rank>";
            output = output + "<title>" + temp.getChild("title").getText() +
"</title>";
            output = output + "<url>" + temp.getChild("url").getText().replaceAll("&",
"&amp;") + "</url>";
            output = output + "</result>";
        }
        output = output + "</search>";
    }
    catch (Exception e) {
        return e.toString();
    }
    return output;
}

```

If we compare this with a translator for a search in Google, the type of difference required between the two applications can be seen:

```

public String doSearch(String SearchTerm){
    GoogleSearch search = new GoogleSearch();
    // Set mandatory attributes
    search.setKey(searchConfig.getGoogleApiKey());
    search.setQueryString(SearchTerm);
    System.out.println("here1");
    // Set optional attributes
    search.setSafeSearch(true);
    search.setProxyHost(searchConfig.getProxyHost());
    search.setProxyPort(Integer.parseInt(searchConfig.getProxyPort()));
    // Invoke the actual search
    return formatGoogleResults(search.doSearch());
}

private String formatGoogleResults(GoogleSearchResult in){
    String output = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><search>";
    try {
        GoogleSearchResultElement[] results = in.getResultElements();
        for (int i = 0; i < results.length; i++) {
            output = output + "<result>";
            output = output + "<rank>" + (i + 1) + "</rank>";
            output = output + "<title>" + results[i].getTitle() + "</title>";
            output = output + "<url>" + results[i].getURL().replaceAll("&", "&amp;") +
"</url>";
            output = output + "</result>";
        }
        output = output + "</search>";
    }
}

```

```

    catch (Exception e) {
        return e.toString();
    }
    return output;
}

```

While there are considerable similarities, the specific calls and structure for each application need to be translated in to the generic service description. The public method descriptions (method name, input and output parameters and types) must remain the same, whichever actual service is being connected to, since these are the methods the broker will connect to; so these represent the generic service descriptions. In the example code, the broker is expecting the search results back in a particular XML format. However, it is inside these methods where the code becomes specific for the actual service used. For the Knowledge Network search, the service is called using a web service call, the results from this service are then transformed to the XML format the broker is expecting (using the private method `formatKNResults`). The Google search differs in that it is called using a Java API, but again the search results need to be transformed to the XML format the broker is expecting (using `formatGoogleResults`). So the type of API the external service uses is not important - Java, web service, direct database connection etc - providing the translator can convert the call from the broker to the required call to the external service, and the results from the external service can be converted back to the format the broker is expecting.

For existing services then it is simply a matter of creating a new translator for any specific application. Obviously if the translator has already been developed for that application then any other user can simply adopt it, so for common applications a library will quickly be established that removes the necessity for any development.

However, for new services, for example `eportfolio`, where there is no current generic service description the process is a little more complicated. Firstly it is necessary to develop the generic service description. This can only be achieved by a survey of different instances of that service to determine a generic set of functions. These need to be coded in a manner that a learning design can utilize. It is then necessary for the generic service description to be incorporated in to the service broker so it can coordinate the services.

This is undoubtedly more work, but it demonstrates the benefits of the open architecture. It is likely that multiple users will wish to integrate a service, and many will be using different applications to realize that service. Instead of each of these users creating an application specific interface a single generic description can be developed, and then only the relatively small task of creating the translators falls to each of the separate parties.

Discussion

In our work we sought to provide a usable Learning Design system by further developing the successful CopperCore engine and the initial work done on the SLeD player. We also sought to gain further insights into the more general issue as to the practicality of the generic service approach, as this had generated a good deal of interest but with few practical examples.

The architecture and methodology we developed represents a compromise between the pure generic service solution and the application specific approach. The use of generic service descriptions by the service broker creates an open architecture where any service can be replaced and added. However, it still requires an element of application-specific coding in the form of the translators. This could be viewed as a temporary measure, since if all the instances of a particular service complied with the generic service description there would be no need for the translators. Such compliance remains unlikely, and so the method outlined here represents a reasonable compromise between the abstract ideal and the market reality.

Although this paper has offered a potential model for incorporating the generic service approach into the current environment, there remain a number of unanswered questions. Firstly, we have not yet determined the efficiency of such a system and whether there is a significant load in transfer of data. Secondly, and perhaps more significantly, we have not explored the limitations of generic service approach. While it is possible to derive a list of generic functions from a range of tools providing the same service, by necessity this ignores differences between them. Thus any particular richness or subtle nuances of a specific program will be lost through this approach since only the generic services are required. This may be the price that is paid for any reusability. A specific instance can always provide a richer example than one which is created to be reused in multiple contexts. The issue is whether that additional richness is worth the cost.

Further work is required to extend the range of services that can be included, and thus to provide a more robust test of the methodology. In addition there are issues which this project did not address, for instance the user interface of different systems, and the extent to which the player, or the initiating service controls this. While it may be possible for the player to offer a uniform user interface by simply taking data in a web services format, this underestimates the significance of how the interface affects the user's behaviour in the originating service. It may be that a software system with a different interface simply does not make sense to the user, or more likely, that it subtly alters their interaction with the system, so that users of the original program, and users of the Learning Design player version behave differently. There is still comparatively little work on the affordances of software and the subtle influences this can have on how a user interacts with a system, but in the general shift towards service oriented architectures with their emphasis on underlying system functionality, it is important we do not overlook the nuances of interface design.

If we return to the fundamental question of this paper, we asked whether the generic service approach had the capability to be a form of universal acid, or whether it was really more of an academic construct destined never to remain the focus of an interested minority. The SLeD project has not provided us with a definitive answer it seems. The architecture and model suggests a way in which the approach could be realized, but the necessity of the translator elements compromises the purity of the solution to an extent, although it also offers a means of bridging the gap between the two viewpoints.

Acknowledgements

The authors would like to acknowledge the invaluable work of the partners in the SLeD project at the Open University Netherlands, in particular Rob Koper, Rob Nadolski, Hubert Vogten and Harrie Martens.

References

- Dalziel, J. (2005). *From re-usable e-learning content to re-usable learning designs: Lessons from LAMS*, retrieved October 31, 2005 from <http://www.lamsfoundation.org/CD/html/resources/whitepapers/Dalziel.LAMS.doc>.
- Dennet, D. C. (1995) *Darwin's Dangerous Idea: evolution and the meanings of life*, New York, USA: Touchstone.
- IMS (2003). *IMS Learning Design Best Practice and Implementation Guide - Version 1.0*, retrieved October 31, 2005 from http://www.imsglobal.org/learningdesign/ldv1p0/imsld_bestv1p0.html.
- Koper, R., & Olivier, B. (2004). 'Representing the Learning Design of Units of Learning'. *Educational Technology & Society*, 7 (3), 97-111.
- Kraan, W (2004). *CETIS Quarterly Newsletter*, CETIS, October 8, 2004.
- Laurillard, D., & McAndrew, P. (2003). Reusable educational software: a basis for generic e-learning tasks. In Littlejohn, A. (Ed.), *Reusing Online Resources: A Sustainable Approach to eLearning*, London: RoutledgeFarmer.
- LeAP Project Case Study (2004). *Implementing Web Services in an Education Environment*, retrieved October 31, 2005 from <http://www.education.tas.gov.au/admin/ict/projects/imsdoecasestudy/LeAPProjectCase.pdf>.
- McAndrew, P., & Weller, M. (2005). Applying Learning Design to Supported Open Learning. In Koper, R. & Tattersall, C. (Eds.), *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*, Heidelberg: Springer, 281-290.
- Weller, M. J. (2004). Learning objects and the e-learning cost dilemma. *Open Learning*, 19 (3), 293-302.
- Wilson, S., Olivier, B., Jeyes, S., Powell, A., & Franklin, T. (2004). *A Technical Framework to Support e-Learning*, retrieved October 31, 2005 from http://www.jisc.ac.uk/uploaded_documents/Technical%20Framework%20feb04.doc.