iLearn – an architectural proposal

Ian Sommerville

This is a short document that outlines a possible architecture for the iLearn DLE. Whilst our remit is requirements rather than system design, the reality of system engineering is that requirements and architecture are inseparable - and attempts to do so, usually end badly.

I use the term “iLearn” in this document to refer to the set of applications and services for learning support which will replace the current DLE. This includes productivity tools such as Office 365 or an alternative system with comparable functionality. The option of providing few or no applications and services centrally, and leaving choices mostly or entirely to local decisions is not precluded and is possible within the architecture proposed here.

The architectural proposals here are driven by 3 requirements:

1. The architecture must be supportive of *incremental development and delivery* of the system
2. The architecture must allow for the inclusion of new services/applications as they become available and for the replacement of existing services (e.g. for contractual reasons or lack of use)
3. The architecture must support 'budget-driven' development i.e. it should be possible to create different instantiations of the system, depending on the budget available.

The fundamental architectural abstraction that I propose should be used is the notion of a *service*. From a user-perspective, services can range from relatively simple, single function services to much more extensive, multi-featured services. For example, an authentication service does one thing only - it simply allows a user to authenticate themselves with a system; a social network for education such as Edmodo is a multi-featured service which does many different things.

The key characteristic of a service from the point of view of the system architecture is that it is a replaceable system component. This means that we can identify abstract services that might be included in the iLearn system such as a messaging service, a coursework management service, etc. without specifying the specific applications/tools that will provide that service.

It also means that if users are unhappy with a particular instantiation of a service, they may be able to replace this with an alternative in their own instantiation of the system. The degree of lock-in to specific providers is limited - when a contract

runs out, a service can be replaced with another if that is the most appropriate action to take.

By adopting a replaceable service approach, we have the ability to create a range of different systems, depending on identified user needs and requirements. The spectrum of systems can include:

1. A completely open system with no centralized authentication, with the only system service being a configuration service to define the independent application services seen by users and groups for cooperative work.
2. A “Glew-like” system with an authentication service and various independent tools.
3. A system with a number of centralized services (authentication, storage, etc.) that facilitate cooperative work.
4. A “system with a fixed set of tools and components.

Aside 1: What exactly is allowed here depends on the governance of the system. I am suggesting that we should have a technical solution, which allows for different options ***not*** *proposing any particular option.*

Aside 2 (for techies): A service here is an abstraction rather than an implementation concept. I am not suggesting that the system implementation needs to be implemented using a web-service model (although it could be).

The service model has an important implication for the overall architecture of the system. To allow for replaceability, then each service has to manage its own data - which can either be done within a system-provided storage service or separately. Some data integration is possible if a shared storage service is used but the notion of tight data integration, a common *data schema* for all applications and maintaining only a single 'golden' copy of information is not possible. Whilst there are benefits in some circumstances to a tightly integrated, database centric system, such systems are very expensive to design (lots of up front agreement on data schemas is required) and difficult and expensive to change. The type of application used in education does not, in my view, seem to require such tight integration.

I suggest that two types of service should be supported in the system:

1. Integrated services. These are services which offer an API (application programming interface) and which can be accessed by other services through that API. Direct service to service communication is therefore possible. An authentication service is an example of an integrated service - rather than use their own authentication mechanisms, an authentication service may be called on by other services to authenticate users; similarly, if users are already authenticated using one service, then it may pass authentication information directly to another service via an API with no need for the user to re-authenticate themselves. Integrated services may be

provided by different vendors and may operate on different platforms - there is no notion that these are 'all in one place'.

2. Independent services. These are services which are simply accessed through a browser interface and which operate independently of other services. Information can only be shared with other services through explicit user actions such as copy and paste; re-authentication may be required. An example of an independent service might be Edmodo - a social network for education.

Of course, independent services may become integrated services if this is seen to be useful, if contractual agreement with the service provider can be agreed and if budget is available.

Aside: There may also be the notion of 'service packages' which are themselves integrated to some extent and which share information. So, MS Office 365 is a service package - the package envelope rather than individual components communicates with the authentication service and manages the 'single sign-on' using its own mechanisms. Google Apps would be a comparable service package and may work in the same way.

Architectural model

An architectural model for the iLearn system is shown in the diagram below.

The elements of this model are:

1. Utility services. These are services that provide functionality that may be required by a number of other services. Authentication, authorisation and storage are obvious examples here but there may well be other common services that will be included. Utility services are integrated services.
2. Productivity package: This will offer services such as word processing, spreadsheets, email, etc. Currently MS Office 365.
3. Application services: These are services that offer specific application functionality e.g. a VLE to manage student work, a specific art package for primary school children, etc. Application services may be integrated or independent services and may or may not make use of the utility services in the system. What application services should be provided initially is an issue for the group looking at the user requirements for iLearn.

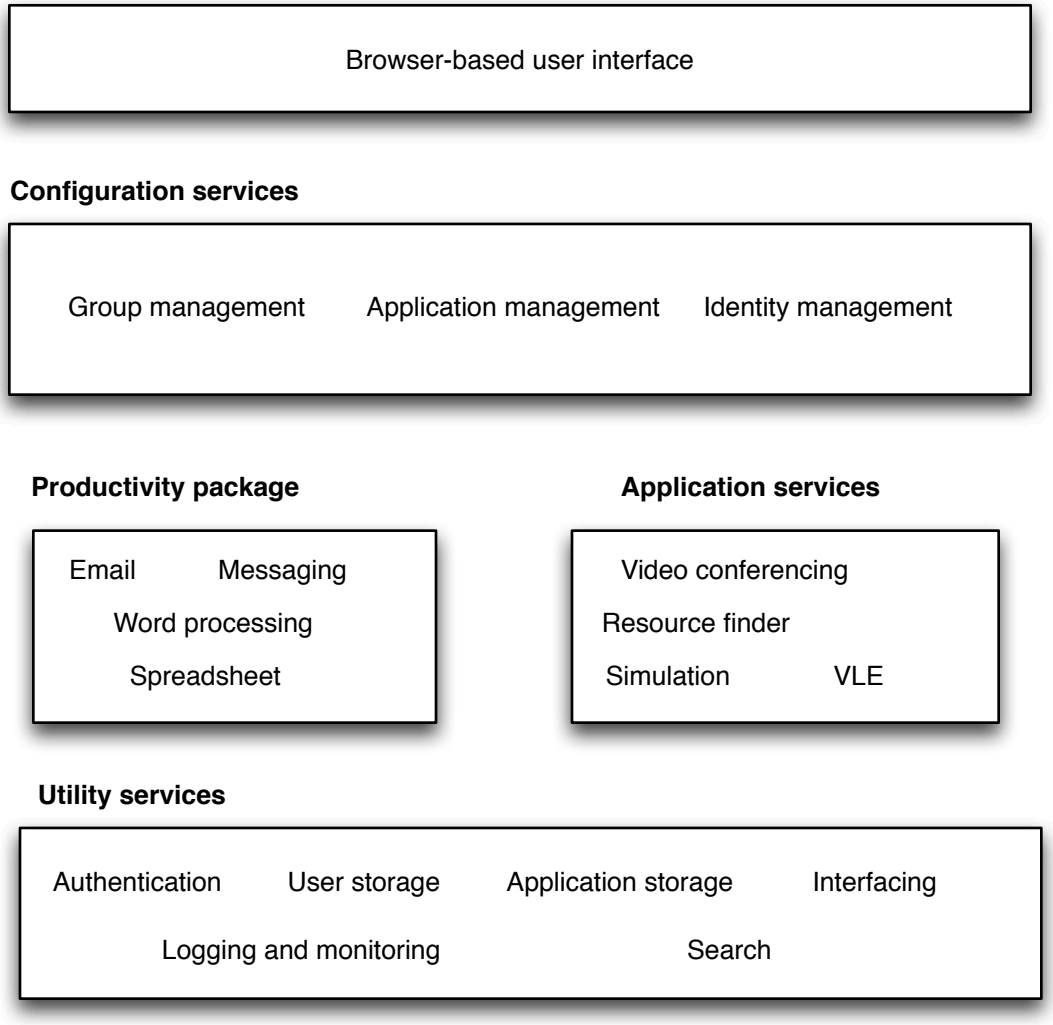

Diagram notes:

All proposed services are EXAMPLES not definitive proposals. What is actually included is up to the people configuring the system.

I am not an ICT in Education expert so what I have identified as application services may be inappropriate –it is up to teaching professionals what is included here.

To allow access from multiple devices, it makes sense to use a cloud-based storage system for user data. However, for regulatory reasons, it may be that some application storage has to be separate and maintained in the UK. Hence the identification of 2 storage services.

Interfacing services at the utility level are used to interface with external applications which may provide data such as SEEMIS.

4. Configuration services: These are services that allow the environment to be adapted for specific groups of users. They should provide they ability to define and manage cooperating groups, to create user interfaces that offer a specific set of services that is appropriate to the class of user and to limit access to system functionality where this is appropriate to do so. Configuration services are integrated services. The notion of a configuration service also means that different local authority policies may be supported.
5. User-interface: A browser-based user interface to access all services. On startup, users will see in their interface the services, documents and groups which are appropriate for their use – e.g. a primary school child will see something quite different form a sixth-former.

   Each service will, of course, have its own interface – there is no notion of implementing a common look and feel interface for all services.

Meeting the requirements.

This service-oriented architecture meets the requirements identified above in the following ways:

1. Incremental delivery. An initial set of key services can be delivered and augmented with new services over time. The only 'essential' service is a configuration service to define groups, applications and permissions but it is likely that an identity management service, including authentication, would be part of an initial delivery.
2. Inclusion of new services is supported through the notion of independent services. When a new service is discovered, it can easily be added and made widely available by modifying the configuration service.
3. Budget-driven development is possible because the principal effort, after some fundamental utility services have been provided, is in converting independent services to integrated services. The extent to which this happens is a function of the available budget.

Glossary

Application programming interface (API). An interface specification which allows one program to directly access the functionality of another program without going through a user interface. Programs can therefore communicate directly without user intervention.

architectural model. An abstract presentation of the organisation of a system. This need not be the same as the final implementation model for the system but is used as a means to facilitate discussion about the system organisation. A

complete description of a software architecture normally requires several different architectural models showing different aspects of the system.

incremental development and delivery. An approach to software development where the features of the software are not completely specified in advance but rather where groups of features are packed as system 'increments' and these are developed and delivered in sequence. Therefore, critical, widely-used functionality is delivered early; less widely used functionality is included in a later increment. This approach allows for feedback from initial increments to influence later increments and for the system development to start without the overhead of all stakeholders agreeing on everything that has to be included.

service: A stand-alone set of features offered by a software system and presented as a single entity. Note that the same software system may offer different services i.e. different sets of features.