ITEC - WP 08

D8.1 - INITIAL INVESTIGATIVE SUMMARY OF TOOLS AND DEVELOPMENT OF MASHUP CONNECTORS

CONTRACT NO	257566
DATE	31/08/2011
ABSTRACT	<abstract max 70 char.>
AUTHOR, COMPANY	Dai Griffiths, Mark Johnson, Kris Popat, University of Bolton
WORKPACKAGE	WP 08
CONFIDENTIALITY LEVEL1	PU
FILING CODE	ITEC-D8.1_Ttt_V1.Doc
RELATED ITEMS	

DOCUMENT HISTORY

Version	Date	Reason of change	Status	Distribution
V1	13/08/2011	1st draft	Draft	University of Bolton

1 PU = Public

PP = Restricted to other programme participants (including the EC services);

RE = Restricted to a group specified by the Consortium (including the EC services);

CO = Confidential, only for members of the Consortium (including the EC services).

INN - Internal only, only the members of the consortium (excluding the EC services)

V2	31/8/2011	2nd Draft	Official	University of Bolton

Executive summary

The purpose of this document is to report on the progress Work Package 8 has made in investigating tools and establishing "Mashup connectors" to facilitate the broader goals of the iTEC project.

In the course of this first year of the project a decision was made by the project as a whole that the scope of tools and services to be delivered by the iTEC infrastructure would be constrained to those which could be delivered by means of widgets, and this informs the scope of the investigation reported here.

Our investigations into different approaches to the provisioning and 'mashing-up' of common toolkits across diverse learning environments has led us to develop a three-pronged strategy which is set out in the following table.

Strategic goal	WP 8 Achievement (with reference to chapter headings)
Identification, establishment and enhancement of a **connector framework** to address the principle barriers of interoperability between tools and learning platforms (shells) in the various environments of the 'classrooms of the future'.	A **connector framework** has been built around the Apache Wookie Widget Server. Solutions have been found to the interoperability challenges presented by a range of different learning environments for integration. The connector framework has been **tested in Liferay and Moodle**
Development of an **iTEC Application Store** around the connector framework to provide teachers and other stakeholders with a common 'place to go' to discover, instantiate and contribute new tools within their own learning environments.	A **prototype Application Store has been developed** with facilities to identify, tag and search tools which can then be instantiated with a learning environment. A **widget discovery service** has been developed. **Designs for further development of the Application Store** in year 2 of the project have been prepared.
Establishing a **repository of**	Building on prior work, we have developed a

described tools which can be instantiated in a variety of learning environments in ways appropriate to that environment and according to pedagogical principles established through iTEC scenarios.	**methodology for describing the ways in which tools can be matched to pedagogical descriptions** of scenarios and recommended according to the technical capabilities of a particular learning environment.

These achievements map on to the detailed tasks of the workplan, as shown on page 12, Table 2. In this deliverable we describe these achievements, and articulate our plans for future development which include the evolution of the Connector Framework to address issues of large-scale implementation across Europe, the extension of the iTEC Application Store into a 'YouTube' for Educational Widgets and the provisioning of tools for teacher-led widget development.

TABLE OF CONTENTS

Reminder of the context

ITEC seeks to transform teaching practice in schools through the development and implementation of innovative pedagogical scenarios which transcend the organisational barriers of existing learning technologies. Principal amongst these barriers is the interoperability between tools across different learning platforms and in the different contexts within which learners learn. Work Package 8 is principally concerned with these issues of interoperability and the consequent needs for tool description and discoverability in the 'classroom of the future' where educational online tools exist independently of learning platforms. Work Package 8's technical solution is to implement an Educational Application Store based on the W3C widget specification and the Apache Wookie Widget Server. Most broadly however, this technical work can be situated as a continuation of the efforts within Learning Technology to establish the means for increasing personalisation and technological flexibility in education.

Purpose and scope of the task

The overall task of WP8, as described in the Description of Work is to

> develop the Wookie Widget server technology to provide a technological infrastructure which supports the mash-up and inter-operation between different tools and services in order to ensure a seamless experience for teachers, learners and other stakeholders while providing the user with access to a variety of tools and services.

Within this context, deliverable D8.1 is described as follows:

> The initial investigative summary will highlight current practice and identify the scope for innovation. Using qualitative measures, focus will be drawn to those tools and services which are deemed likely to have the maximum impact in the first year. In parallel to the investigation, the infrastructure for integrating Wookie into existing classroom technologies (IWBs, VLEs, etc), building facilities for Widget interoperability and allowing for the tagging and discoverability of Widgets in the server will be developed. This will consolidate the functionality of Wookie in the school environment, and create a space for early adoption and experimentation amongst teachers.

The iTEC workplan, and the description of the tasks for WP8, make clear the central role of widget technology in providing adaptable activities across a range of target infrastructures, and the leading role of Apache Wookie in that work. Consequently a principal focus of our investigation has been the ways in which widget technology can meet the needs of the project, and the degree to which it does this. The following table indicates how Wookie meets the characteristics of the target technology abstracted from the DOW:

Table 1: Wookie Capabilities

Characteristic needed	Wookie capabilities
Functions with a range of infrastructures	Normally tools and services have to be integrated individually into each platform. This is impracticable for iTEC. This is resolved by a single integration of Wookie into a platform, which gives access to the whole range of widgets stored on the server.
Supports the pedagogic adaptation of scenarios	Wookie can store a large number of tools and services. These are all available for all supported platforms. Support for scenarios can be localised by selecting the most appropriate of these.
Enhances the management of teaching	A long standing critique of e-learning scenarios and learning designs is that they force the teacher to follow a set path. Wookie enables teachers to easily change the tools which they make available, should the dynamics of the class require a change
Centrally managed	Wookie enables tools and services to be curated centrally. This could, for example, be at the level of Europe, Education Ministry, or school. The range of widgets available to teachers can be defined, and categorised.
Flexible	Wookie brings web-based tools ready-to-hand for teachers. Because of this, it facilitates exploration and improvisation in the use of a variety of different tools, enabling teachers to easily experiment with different technical options. The range of tools available and the ease with which they can be created and shared opens the door to a level of technical engagement by teachers beyond the existing LMS.

Once the capabilities of this technology became clear, a decision was made at the level of the project technical committee that the scope of tools and services to be delivered by the iTEC infrastructure would be constrained to those which could be delivered by means of widgets. Their role in the project at a technical level is made explicit by the architectural diagrams contained in deliverable 9.1.

As a result WP8 has prioritized the strategic goals identified in the executive summary, which enable us to meet the goal identified in this deliverable description above: *consolidate the functionality of Wookie in the school environment, and create a space for early adoption and experimentation amongst teachers*.

The work carried out can all be ascribed to one or another of the tasks defined for the work package, but a task centred view on the work carried out would not provide a coherent picture. Nevertheless, for those readers who would like to relate the contribution of WP8 to the tasks set out in the Description of Work, Table 1 (page 12) will clarify the relationship.

Relationship with other tasks

Work Package 8 plays a central role within the technical infrastructure of iTEC and consequently has a range of relationships with other work packages and tasks.

These relationships can be situated against the three principal areas of activity:

a) Connector Framework: the future users of the connector framework in iTEC will be WP7, who will use it as the enabling technology in their development of the iTEC composer.

b) iTEC Application Store: the leveraging of the Wookie Widget Server as a way of bringing to-hand a rich and diverse range of W3C widget tools which are appropriately tagged has a key bearing on those work-packages which seek to make the connection between the specification of scenarios and the performance of activities with tools within those scenarios. The Application Store and the description of tools has drawn extensively on the work of Work Package 3 in the identification of specific activities within scenario contexts.

c) iTEC Widget Repository: distinct from the more generic Application Store, this is intended to integrate with the iTEC Registry in Work Package 9 and the iTEC Scenario Development Environment in Work Package 10. Much work in Work Package 8 has involved creating a common framework for the description of tools such that descriptions stored in the iTEC registry, which are used for making recommendations for tools within the technical settings of schools, are consistent. The iTEC registry created by WP9, and the information handled by WP10, also refers to entities other than widgets, and makes recommendations on the basis of this information, although Work Package 8's affordance approach can equally apply to these entities.

Structure of the document

This report is structured around those three aspects of our work, with a chapter devoted to each:

- Providing Mashup Connectors to Existing Components (Section 1, Page 15)
- The iTEC Application store (Section 2, Page 29)
- Components of the iTEC Widget Repository for Scenario Instantiation (Section 3, Page 39)

In addition to a section on Specification and Standardisation (Section 4, Page 50), technical details of the work of WP8 are provided in the Appendices.

Impacts of Work Package 8

Milestone and Task Review

The achievements of Work Package 8 are documented in the table below:

Table 2: Relation of work carried out to project tasks

Relation of work carried out to project tasks		
Task No.	**Short description**	**Achievements**
T8.1 (m 1-36)	Investigation into tools/services and plug-ins of interest	The potential of widget technology in meeting the requirements of the project has been extensively explored, and the results of this analysis are manifested in analysis and design of the systems and plans presented in this deliverable. A new plug-in has been developed for Liferay, and work has been carried out to further develop an existing Moodle plug-in (See **Page 22, Section 1.7 "Connector Framework and iTEC shells"** for details of these plug-ins)
T8.2 (m 1-12)	Development of mash-up connectors and adaptations of existing tools, services, and plug-ins	This was a principal focus of work, as this is the only task which ends in year one. Once a stable release of Apache Wookie had been achieved the connector framework was developed, and is currently being deployed. See **Section 2, Pages 29-38** of this deliverable for details.
T8.3 (m3-24)	Development of missing components	"Missing" is understood to mean that there is an unmet requirement for a functionality in a scenario. Work has begun on a spreadsheet widget in response to this (see **Section 3.4, Page 45 "Widget Provisioning and Scenarios"**), and it is expected that more requests will emerge in the coming year. A strategy has been established for widget harvesting and creation, and a template based tool has been planned.
TD.5 (m1-18)	Task 8.5 Scenario and widget integration and testing	WP8 has maintained collaboration with WP3, which is developing the prototype scenarios, particularly in the identification of tasks within scenarios and the identification of the affordances of tools. As scenarios become more mature integration testing will increase in intensity.

T8.6 (m8-48)	Maintenance	A Wookie server has been established, configured and maintained at Schoolnet available at http://itec-wookie.eun.org/wookie
Task 8.7 (m1-48)	Information modelling of services/tools and plug-ins	In close collaboration with WP10 a schema for modeling the functional affordances of widget based services and resources has been established and agreed by the project. See **Section 3.1, Page 40, "Affordances, iCAMP and the description of tools"**
T8.8 (m6-48)	Impact on pre-standardisation	WP8 maintains close contact with W3C. Innovative functionality has been added to Apache Wookie, a reference implementation of W3C widgets. This then becomes a candidate for pre-standardisation. As the project infrastructure matures, proposed extensions and/or changes will be submitted to W3C.

Risk Analysis Review

Most of the risks around Work Package 8 do not concern the work package itself, but the dependencies of other work packages on the outputs for Work Package 8.

The risk concerning "Delays in ontology and formal model provision" was identified in the description of work. In year 1, it became apparent that the coordination of dependent tasks concerning tool description would be a challenge. The role of Work Package 8 in describing widgets and tools has created dependencies between WP8 and WP9 and 10. The combination of the complexity of the task of tool description together with needs of WP9 and 10 to agree datamodels early on in the project contributed to initial difficulties, which although did not affect WP8 itself, did affect Work Packages 9 and 10. The presence of the iCAMP work on tool description has eventually addressed these problems. It is worth mentioning that the ability to *demonstrate* the iCAMP concepts with *working software* was an important step forwards for all three work packages.

The capabilities of Wookie may mitigate some of the risks concerning the diversity of technical settings in the project or the possibility of shells becoming obsolete. The flexibility of Wookie and the simplicity of its plugin architecture should enable it to 'keep up' with the changing technological landscape.

The risk of "too little interaction between developers of scenarios and developers of the technical platform" is likely to remain a challenge for WP8 as for other work packages. The provision of a wide variety of widgets on the iTEC Application Store, together with close engagement with Interactive Whiteboard providers should provide some technical 'outreach' to the pedagogical Work Packages.

Ethical Issues

No ethical issues have arisen during the course of work for Work Package 8.

IPR issues

Wookie software developed in Work Package 8 is available under the GPL and made available through the Apache foundation.

1. PROVIDING MASH-UP CONNECTORS TO EXISTING COMPONENTS

1.1 The role of connectors within iTEC

There are major technical barriers to developing new teaching practices with technology. For example, currently it is not possible to:

a. Use the same tools across different learning environments.
b. Integrate activities between different tools in different environments

This deficiency in current learning technology provision has been known for many years. Liber and Britain, in their influential report on Virtual Learning Environments in Universities (1999), identified the deficiencies in Virtual Learning Environments with regard to their tools being locked-in not just to the particular VLE platform, but with little provision for tools to be deployed across modules or lessons in ways which would facilitate innovative pedagogical organisation. Whilst aiming their criticism at VLEs in Universities, the criticisms are equally applicable to the school curriculum, with the ability to work with technologies across the curriculum meeting with the same interoperability barriers. Liber and Britain point out the opportunity within e-learning which this presents:

> *"This could be a place where eLearning tools could offer new opportunities, by offering new frameworks to allow coordination between multiple modules' activities; however most leading VLEs do little to support learning at the programme level."*

Meeting this requirement is the focus of our establishment of a 'Connector Framework' within the iTEC project.

The characteristics of the connector framework are that it should be:

1. Adaptable, so that it is capable of
   a. functioning with a range of infrastructures in different schools and countries
   b. supporting the pedagogic adaptation of scenarios for differing school contexts
2. An enhancement of the ability of teachers and educational leaders to manage the teaching for which they are responsible
3. Capable of being centrally managed, so that the coherence of the pedagogic designs and technical offering is maintained

The connector framework addresses the specific issues of interoperability between tools and platforms and the removal of technical barriers. This interoperability work is the principle foundation upon which the rest of WorkPackage 8's has proceeded: to address the need to establish a platform from which the technological practices of teachers can be developed through creating an 'AppStore' of educational widgets; and to provide a repository of widgets which integrate with iTEC pedagogical scenarios.

The iTEC project plan identifies emerging Widget technologies as the means whereby this challenge can be addressed by the project. This is achieved by building on the Apache Wookie widget server, which itself originated in the FP6 project TENCompetence. The DOW requires the WP to:

> *develop the Wookie Widget server technology to provide a technological infrastructure which supports the mash-up and inter-operation between different tools and services in order to ensure a seamless experience for teachers, learners and other stakeholders while providing the user with access to a variety of tools and services.*

This approach has been confirmed by the project during its first year of operation.

The way in which the underlying Wookie Widget server can transform the functioning of existing LMS technology is represented in Figure 1 below. Drawing on Koper's[2] distinctions an LMS is represented as a system which brings together an *Environment* of tools and services and resources, and *People* (teachers and learners) where teachers can organise learners into learning activities with those tools and resources.

At A), a number of LMS systems are represented, each having their own tools and resources, and managing the people within their contexts. With this set-up, there is no scope for coordination of activities across different platforms (which might mean coordination across different departments, different institutions, etc), because there is no coordination between the tools in one LMS and another

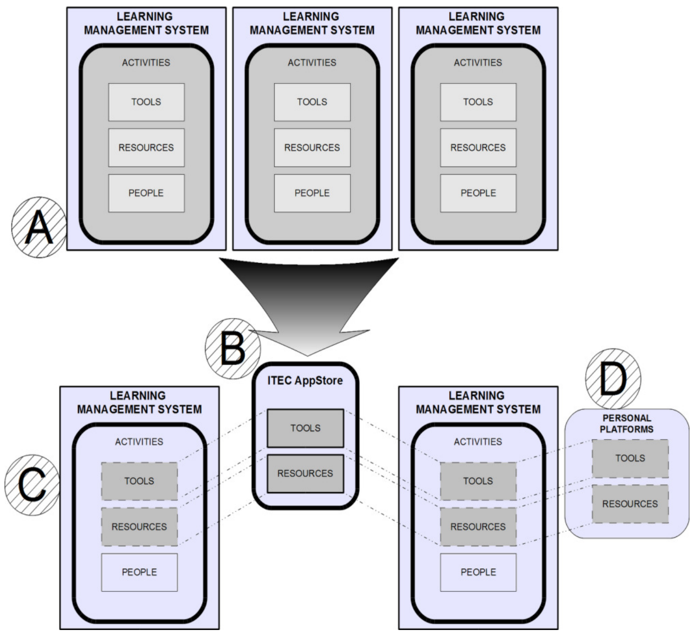


Figure 1:Wookie and Mashup Connectors

[2] Koper, R (2004) Educational modelling language: modelling reusable, interoperable, rich and personalised units of learning *British Journal of Educational Technology, vol 35, no. 5*

The Wookie/iTEC intervention is to transform this situation by implementing an Application Store (AppStore) of widgets (B). Using this approach, and expoiting the fact that tools and resources in the iTEC Application Store can be 'plugged in' to existing LMSs in other contexts creates the situation where the tools and resources that were once contained and coordinated within those LMSs are now contained within each LMS (C) (by virtue of a plugin), but potentially can be coordinated by the Wookie technology (B).

This allows for mashup connections in a number of ways.

1. The connection between different platforms (potentially in different institutions or learning environments)
2. The connection between institutional and personal technology (D) where institutional tools and resources may be embedded in learners and teachers personal tools (for example, a mobile phone)
3. The connections between different tools coordinated within the iTEC appstore through inter-widget communications

1.2 Components of the connector framework

A 'connector framework' is a broad term for a set of APIs (services) which allow for the instantiation of and communication between a common toolkit across a range of different platforms. Such toolkits, including Google OpenSocial Apps and 'gadgets', are increasing in popularity and are featuring strongly in efforts to realize learning environments which marry centrally-provided tools with Personal Learning Environments (see, for example, the EU-funded ROLE project).

In these projects, efforts have been made to facilitate the interoperation of widgets across the diversity of platforms that they might be engaged with, removing barriers of authentication, data sharing and platform dependence. Typically, developers can create plugins in new environments which allow for the linkage (including user authentication) and embedding of tools. RESTian apis provide function calls to allow the plugin to get lists of widgets/tools, set user information, instantiate widgets or get a url to retrieve a widget. The essence of this approach to a 'connector framework' is shown in Figure 2, where API calls are provided both to instantiate tools and to manage users.

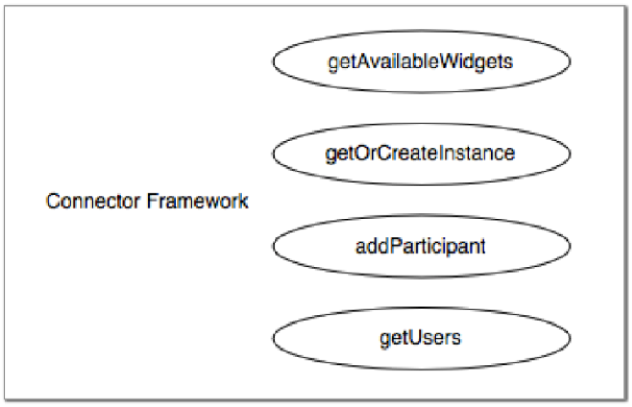


Figure 2: The Connector Framework API

1.3 Apache Wookie as a Connector Framework

As a server-side support and delivery mechanism for W3C widgets, with additional support for Open Social gadgets and widgets with specific Wookie features, Apache Wookie provides a key example of a ‘connector framework’. Wookie functions to both store and deliver W3C widgets to a range of platforms through the provision of the connector framework API. Wookie manages the unpacking and delivery of widgets to web applications and download to devices that already support widget packages. It acts as a mechanism for managing widget users and facilitating data storage and widget interoperability.

Wookie mechanisms allows for a rich set of additional tools and content (in fact anything that can be housed in a browser) to be integrated with existing shells such as virtual learning environments, social software, mobile devices and whiteboards. The technical challenge for this work-package lies in taking a technology designed for delivering small, self-contained applications and allowing them to be collected, connected (mashed-up) and delivered to the specific shell requirements of iTEC. Unlike other widget platforms (for example Google gadgets), Wookie is platform neutral, requiring for authentication purposes only a ‘screen name’ of a user, which is passed to it from a shell. As a result, collaborative multi-user activities can be established in Wookie with no need to create users for that particular activity. In effect, this means that the user management for a Wookie widget-based activity need only be done by the shell that instantiates the widget, thus removing one of the principle barriers to the integration of external tools.

1.4 Connector Framework Main API

Each individual plugin whilst making use of the Wookie connector framework is its own entity and is more akin to the environment that it sits in than it is to Wookie. For instance, in the case of Liferay the plugin has been written as a Portlet using the JSR 286 (ref) specification, it is written in Java. This means that it should work with any environment that supports JSR286. For iTEC it has been specifically targeted and tested on Liferay. The current version of the plugin can be found at:

(http://iecbolton.jira.com/svn/ITEC/liferay_plugin/trunk/). The Moodle plugin has been written as a Moodle block in php. It can be found at:(https://github.com/krispopat/Wookie-Moodle-Connector). The source code for the connector framework is part of Apache Wookie which can be found here: `http://svn.apache.org/repos/asf/incubator/wookie/trunk`

The user experience in each plugin is, however, similar. There are three different modes of use. The first mode is the configuration mode. In this the plugin is configured with a host url for Wookie itself and an API key. This key is created within Wookie to identify the calling environment. It is used by Wookie for data sharing which is particularly useful for widgets that need data to be persisted and communicated between users or for persisting data for a single user. For instance, a chat widget or a vote widget send the chat or vote data to the server, using Wookie's "sharedDataForKey" function this is then accessible to other users of the same widget given that the widget id and the API key are the same.

1.5 Connectors between Wookie and iTEC target platforms

The mechanism that allows widgets to be integrated within shells is the connector framework. For each shell a plugin is written in the language of the shell allowing widgets to be embedded. All shells can make use of the same Wookie server, therefore allowing: a common set of widgets, communication between shells and a cross shell understanding of identity. The following diagram (Figure 3) gives a simplified overview of this system.

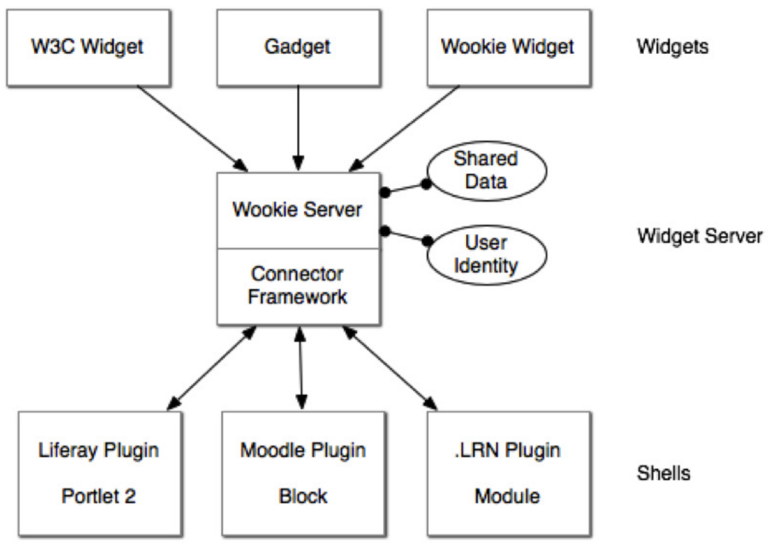


Figure 3: Architecture of the Wookie Connector Framework

As can be seen Wookie currently supports three different types of widget. Wookie Widgets are basically W3C widgets with some additional features allowing data to be shared and user identity to be understood. Gadgets are Google Gadgets and OpenSocial applications. Wookie has a converter built in, which translates these to W3C widgets before delivery. Consequently, the connector framework is not concerned with the specifics of the widget being delivered, and neither is the shell.

The shells shown in the lower part of Figure 3 are the shells adopted for iTEC development in the first year of work. The connector framework has been tested by WP8 in Moodle (building on an existing initial implementation) and Liferay (developing an entirely new connector).

1.5.1 Development of a stable platform for the connector

The Apache foundation has provided a platform from where the stability and robustness of Wookie can be tested and improved. Development work on iTEC has produced the first stable release of Wookie within Apache. The release is in two versions, one is a source release and is designed for developers to download via svn, build and run. The other is a binary standalone version designed so administrators can simply download it and run it.

The source release can be found here:

http://svn.apache.org/repos/asf/incubator/wookie/branches/0.9.x/

The binary release will be available on the Wookie web site when the approval process has finished but can be built from the source using the build-release-all target in the ant script. Instructions on how to do this are included with the release.

The version of the server which was developed included improvements to the server which provide essential functiionality for iTEC, in particular a stable persistence system which enables users to be presented with their widgets in the appropriate state when they return to the system.

A middleware persistence manager library called OpenJPA (http://openjpa.apache.org/) was used. It is an open source implementation of the Java Persistence API or JPA the specification of which can be found at http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html. This allows the model of the data being stored to be abstracted from the actual storage mechanism. In effect this means the person running Wookie can choose which database they wish to use. If the user doesn't wish to be concerned with configuring and connecting a database to Wookie it also has an embedded database management system built in and will create and configure it's own tables on installation.

The standalone build was created and tested, so that the server can be used in iTEC sites without the need for a skilled administrator. To enable this all libraries, configurations and folder structures were built into the standalone. This was more than taking a snapshot build from the source release. A new build target needed to be created because the original source only version of Wookie loads the libraries and the application server required on the fly and places them in different places on the host machine. There is also the capability of creating a web archive version of the build which can be deployed under Tomcat or some other application server if the administrator so wishes.

This is an important aspect for iTEC. In the first place it enables the development team who are working on scenarios, narratives and learning stories to install their own Wookie server and develop and manage their own widgets there. Secondly there are national regulations which prohibit the posting of personal data to servers outside their boundaries, and other more restrictive regulations may be applicable to schools. The ability to easily spawn new instances of Wookie in whatever context the regulatory framework requires is therefore essential.

Work has also been done extending the remote administrative capabilites of Wookie. There are now RESTful services which allow the management of the server. This has allowed iTEC to begin developing administrative widgets or extensions to the shell to administer Wookie. The specification for all of Wookie RESTful services can be seen in the appendix.

Release Timescale for Connector Components

The current connectors for Liferay and Moodle are available. They will be maintained as they are with bug fixes as required.

Both connectors will be branched to support the new App Store. As part of the App Store development a new connector framework will be developed upon which the branched connectors will be developed. The following table gives an outline of the App Store release timescale and associated connectors.

Project Month	Release Title	Description
12	Prototype App Store	The app store prototype will be a set of functional html pages which will standalone. It will demonstrate the app store concept and allow the job of uploading and tagging widgets to begin.
15	First Embedded App Store	The app store will be embedded with connectors – new connectors to be release at this point. Widgets will be included. Testing will begin
24	App Store Release	All features of the App Store will be in place after iterative develop/test cycles have taken place.

1.6 Interoperability and Authentication Across Shells

This issue addresses the case of a user who logs into a shell, say Liferay, and uses some widgets in some way, perhaps for chat, for storing images or playing a game, and then the same person – perhaps at a later date – logs into a different shell. It is desirable, when the user logs into second system, that they find that the widgets they are using with the same information and state as had when they were using them in the first system. Effectively the widgets ‘know’ who is looking at them.

Currently the ID of the person who has logged into the shell is passed back to Wookie to identify them. The specifics of that id are in the domain of the plug-in. The plug-in might send the screen name, the id, the first name - anything that they feel is appropriate for view. Currently this is the only way that Wookie knows who the user is. So logging into different shell environments would not necessarily result in the user seeing the same information – or even the same name.

With this in mind the iTEC Application Store (see **Section 2, Page 29**) interface will request a common id from the shell. Initially the plan is for the user’s email address to be utilized as the unique id for that person. So long as the user has inputted the same email address the data associated with that user in any widget storing data in the Wookie backend will be available to them. The choice of a common authentication scheme across the iTEC project will be addressed during the second year of the project. The most likely candidate for this at the moment is oAuth (ref). Whatever solution is adopted by the project will be adopted by the iTEC Application Store. The creation of a Widget Discovery Service (see **Page 31, Section 2.3 “AppStore Widget Discovery Service”**), rather than multiple connector-framework based plug-ins will make the transition to the adopted solution far easier to implement.

1.7 Connector Framework and iTEC Shells

1.7.1 The LifeRay Shell

The Lifray plug-in is a JSR286 standard portlet. In Liferay a Portlet appears as an application a menu to the right, if you have administrative privileges as is shown in the screenshot below.

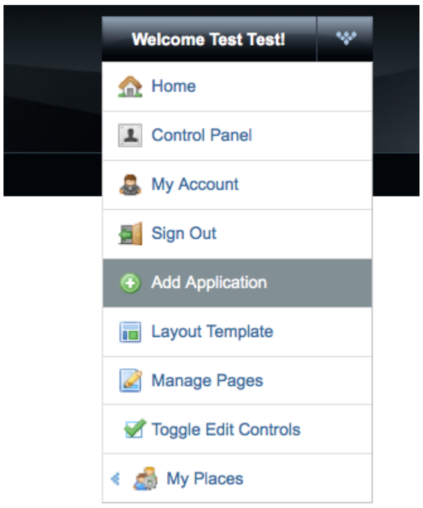


Figure 4: LifeRay adding a new application

When this has been selected the application panel appears. Currently Wookie can be found under the sample category as is shown here.

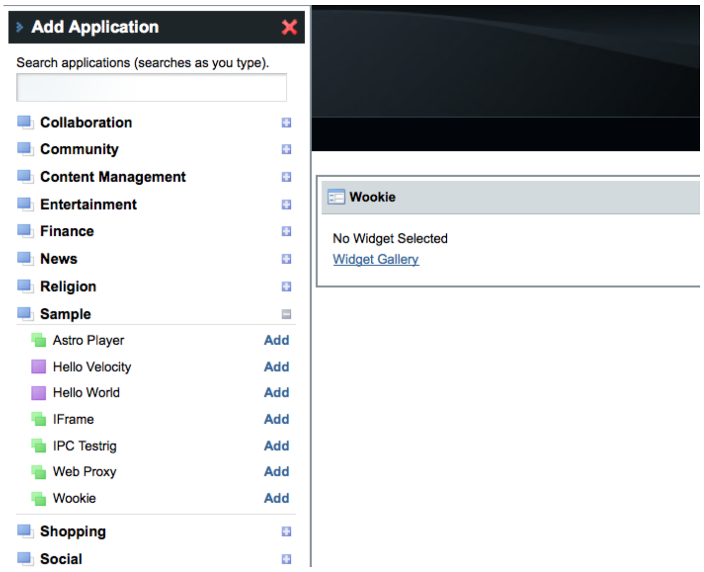
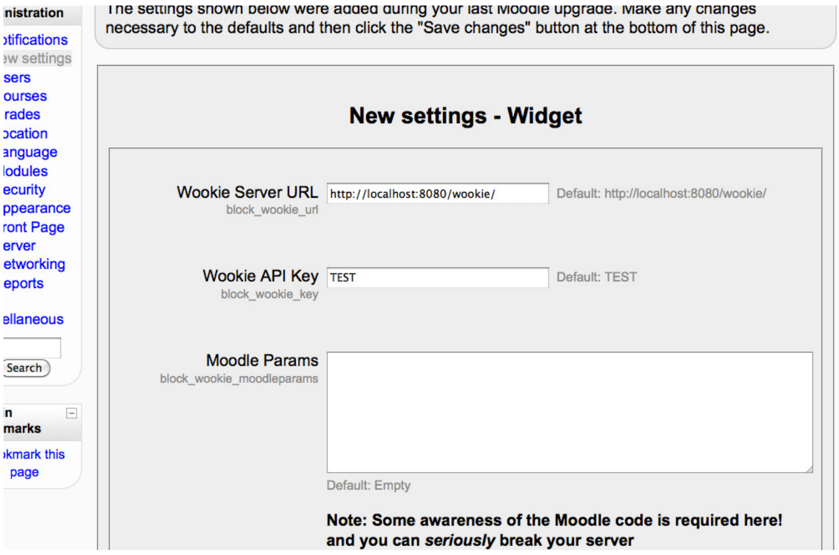


Figure 5: LifeRay Adding Wookie Widget

Then the process is to open the preferences for the portlet and configure it with the required host and API key information, go to the gallery and choose a widget. Screenshots and explainations for this can be found here in the plug-in specification section.

1.7.2 The Moodle Shell

As far as Moodle is concerned the process is slightly different. A moodle block has been written which is dropped into the blocks folder of the moodle installation. After the block is dropped into the blocks folder by clicking notifications and then continue the following configuration screen will appear.



The settings shown below were added during your last Moodle upgrade. Make any changes necessary to the defaults and then click the "Save changes" button at the bottom of this page.

New settings - Widget

Wookie Server URL (block_wookie_url): http://localhost:8080/wookie/ — Default: http://localhost:8080/wookie/

Wookie API Key (block_wookie_key): TEST — Default: TEST

Moodle Params (block_wookie_moodleparams): — Default: Empty

Note: Some awareness of the Moodle code is required here! and you can *seriously* break your server

Figure 6: Moodle establishing link to Wookie Server

Here the required fields are the Wookie server url (the host) and the API key.

1.7.3 Adding a Widget in Moodle

The process of adding a widget in moodle starts by turning editing on then adding a new block from the menu's that appear at the sides which reads "add widget". The Figure 7 shows where a widget can be added.

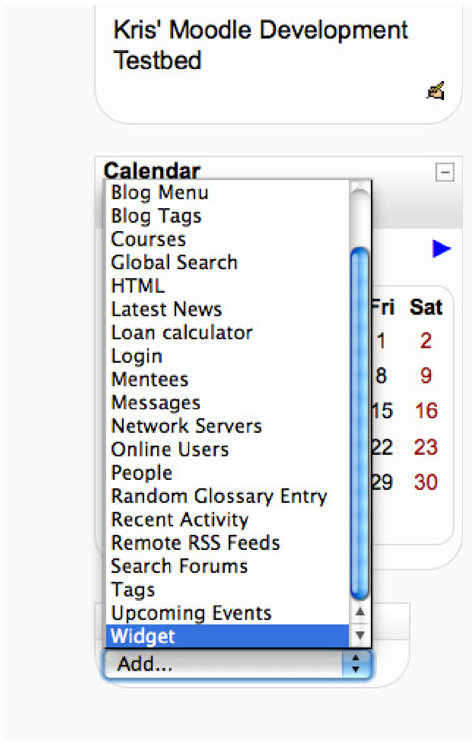
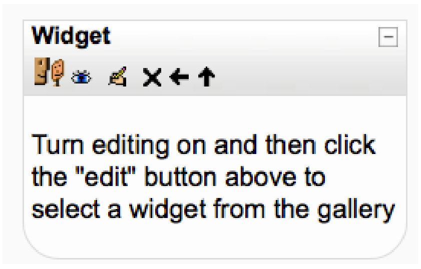


Figure 7: Moodle adding a widget block

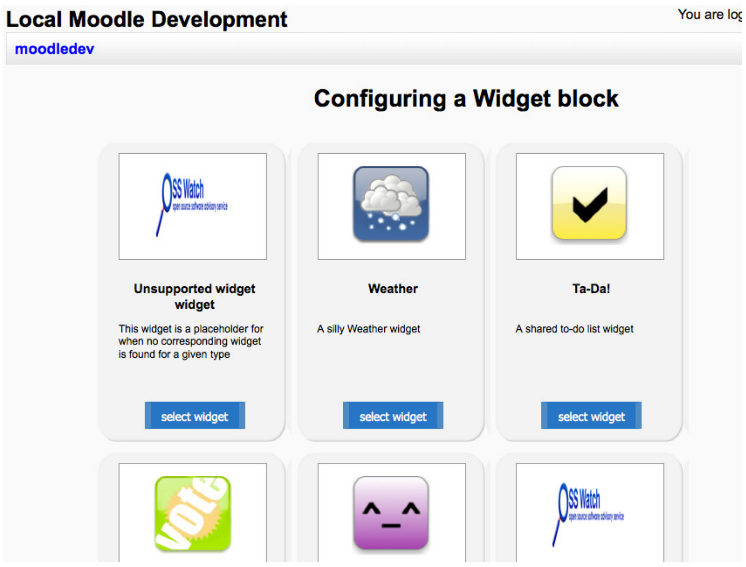
On selecting ‘add widget’, an empty widget container is added to the side-bar in Moodle. This container can then be configured to display a widget of the user’s choice by clicking on the ‘edit’ button (see figure 8).


Widget

Turn editing on and then click the "edit" button above to select a widget from the gallery

Figure 8: Moodle empty widget block

Clicking the edit button in the widget block takes the user to the gallery from where they can choose a widget from the list as displayed in Figure 9.


Local Moodle Development

You are log

moodledev

Configuring a Widget block

Unsupported widget widget

This widget is a placeholder for when no corresponding widget is found for a given type

select widget

Weather

A silly Weather widget

select widget

Ta-Da!

A shared to-do list widget

select widget

Figure 9: Moodle configuring new widget

Once a widget is selected from the gallery it is instantiated within the moodle widget block on the course sidebar.

1.8 Testing and Valorisation of the Connector Framework

1.8.1 Testing the quality of the code within Apache

iTEC has a policy of basing its software on open source applications, and in the case of WP8 this means Apache Wookie and Liferay. The strategy for testing and valorization is, wherever possible, to submit code to the open source project itself, as this ensures a rigorous quality control process and the maximum potential for valorization. This is the case for the Connector Framework, which has been developed as a contribution to the Apache Wookie project itself, and testing of this implementation is being coordinated through the Apache Software Foundation, where a robust process of issue-tracking and bug-fixing is in place.

This approach is only valid if activity on the open source project is at a high level, and if it can be reasonably expected that this will continue in the future. Fortunately Apache Wookie is gaining maturity in a number of different contexts as iTEC progresses. At present a number of HEIs in the UK and the EU are currently exploring Wookie in projects sponsored by JISC and the European Commission (in addition to iTEC). In addition, a small number of UK HEIs have installed Wookie servers with a view to integrating widget functionality into their LMS.

1.8.2 Testing the quality of the code with other users

In addition iTEC is maintaining close contact with other groups who are making use of Wookie, but who are not contributing significantly to the code base. This enables the project to obtain an additional channel feedback on the performance of Wookie.

iTEC staff took part in a Widget dissemination and training event on 23rd – 24th March organized by partner University of Bolton within the CETIS service which they run. http://wiki.cetis.ac.uk/Widgetbash. The stakeholders at this event included University lecturers, software developers, national project coordinators and institutional managers with an interest in deploying Wookie. The 'code-bash' format provided a means whereby this range of stakeholders could 'get their hands dirty' with the widget technology: partly as a way of revealing that the business of creating cool widgets is not very technically demanding. Giving an insight into the ease of technical development was not just useful for developers; the ease of technical development gives managers confidence in deploying the technology, and teachers the confidence that they can develop things themselves.

The team also participated in the JISC CETIS OER code-bash, which provided opportunities for valorization http://wiki.cetis.ac.uk/OER_Hack_Days. In a similar way to the 'Widget Bash', Wookie was experimented with by a range of stakeholders as a potential platform for content and tools relating to Open Educational Resources.

Other activities which are contributing directly both the testing and valorization of the Wookie server include the input of iTEC work to:

- Manchester Metropolitan University are using Wookie in Moodle using the Moodle Plugin developed within iTEC
- The WIDE project for the creation of widgets, funded by JISC in the UK
- The WIDGAT project has recently been funded by JISC for development of widget creation widgets
- The JISC DVLE programme is focused on education, and has the overall aim of finding ways of extending and evolving the functionality of existing LMSs. Among other technologies it is making use of W3C widgets and Wookie.

1.8.3 Testing fitness for purpose

The quality control and bug testing processes of the Apache Foundation will ensure that the code produced by iTEC WP8 is of good quality, and this is confirmed by the feedback obtained by other projects using Wookie. However, while this ensures good quality, it does not guarantee that the code produced meets the requirements of the iTEC project. Consequently a plan for the testing of the connector framework has been specified detailing the:

- integration with existing learning platform architectures (target shells)
- Integration with existing classroom-based technologies (e.g. Interactive Whiteboards)

There is a growing list of learning platforms which are now capable of running Wookie Widgets. The testing of each of these will focus on the functionalities of the Widget interfaces themselves, rather than the specific functionalities of the host platforms (iTEC shells). However, 'environments' consist of a variety of different elements, including Web Browser standards compliance, and the security settings of individual environments.

Each Widget may make different demands of services and HTML standards in different settings. As the number of Widgets available through the iTEC environment increases, each of these widgets will be tested within those different environments. Data concerning the range of technical settings which are possible for the instantiation of different Widgets will be stored in the iTEC registry which contains detailed information about individual tools, in a data list extracted out of engagement with the iCAMP project.

1.8.4 Testing the Liferay demonstrator of the connector framework

Liferay is a very well established product in the corporate market. Gartner reported in 2010 that *Open-source Liferay has made the strongest move toward the forefront of portal* *decisions for Gartner's customers.* http://www.gartner.com/technology/media-products/reprints/liferay/206214.html. The iTEC developed portlet will be made available to the Liferay community, and it may be expected that this will lead to substantial feedback on its effectiveness. However, unlike the work carried out on Apache Wookie, the code developed by iTEC does not contribute to the core code of the Liferay project. Consequently the quality and effectiveness of the solution developed needs to be validated elsewhere. It is planned to achieve this by implementing the solution as a public demonstrator on the iTEC Liferay server which is currently used as the project intranet and dissemination Website. It is anticipated that this will also be used in training within iTEC. A close relationship has been established between the WP8 development team and the administrators of the Liferay server in Schoolnet, and once the connector framework goes live detailed information on its performance will be made available to the development team.

2. THE ITEC APPLICATION STORE

Addressing the interoperability barriers to using tools across environments is essential if any change in practice is to be made possible. However, practice change will not necessarily follow the removal of technical barriers. The need to encourage and inspire teachers with what is possible and with what becomes (following the removal of technical barriers) 'ready-to-hand' for them remains. Teachers are very busy, and often over-burdened with new systems and practices. Thus anything which iTEC does to engage teachers must in some way reduce their technological complexities rather than add to them.

Currently, when seeking new tools or online activities to try with their learners, teachers are faced with a bewildering combination of internet searches, frustrations in installing software, remembering and forgetting URLs, configuring users, hardware, remembering passwords, etc. It is little wonder that only the most intrepid will venture beyond the confines of the immediate affordances of the tools in front of them.

In recent years, however, an alternative to this complexity has emerged with the metaphor of the online 'AppStore'. AppStores provide a single place to go where new functionality, tools, and activities can be added to mobile and desktop devices with a guarantee that they will work without further configuration. However, most of these are currently proprietary systems tied into particular operating system architectures. With the interoperability opportunities presented by Wookie Widgets, an Educational AppStore presents itself as a way of extending the metaphor of 'Apps' into the education space and providing teachers with a solution to the over-burdensome processes of discovering and installing new tools.

Having established a 'connector framework' for technology which allows for the instantiation of tools and services across different platforms, the challenge is to bring this into a form which is usable by teachers in the field. Work Package 8's iTEC Application Store solution means that the practice of searching for, uploading and installing apps is one that can become common across a range of situations. This standard 'place to go' integrates seamlessly with the operating environment of teachers. Thus, for the teacher, the experience of adding tools is made as uniform as possible irrespective of the environment and platform from which they are working.

2.1 AppStore Progress in Year 1

In year 1, in preparation for the establishment of a community-driven Application Store, and as part of the developments around the tagging and searching of widgets, Work Package 8 has sought to enhance the user experience of Wookie so that widgets can be easily searched for and catalogued, browsed and instantiated across a range of different platforms.

2.2 Appstore Interface Design

The following interface designs show the screen layout of the iTEC store.

Colours used are all present in current iTEC website (with the darker shade of purple being an addition)

2.2.1 Design for the main page of the iTEC Application Store

The design of the iTEC AppStore has been created to establish a unified 'look and feel' for the process of searching, browsing and installing iTEC widgets. On adding a Wookie widget to any of the compatible learning platforms (shells), a screen similar to Figure 17 will be displayed, including a categorized list of the 'affordances' of widgets, that is to say, the ways in which they can easily be used. Alternatively, a list of subjects to which certain widgets might be appropriate can be displayed. With the list of widgets displayed against a category, their user ratings will be displayed together with their date of deployment. When a category is selected the widgets will be ordered by their affordance weight value. That is by how much they "afford" activity under that category in the "All widgets" panel.

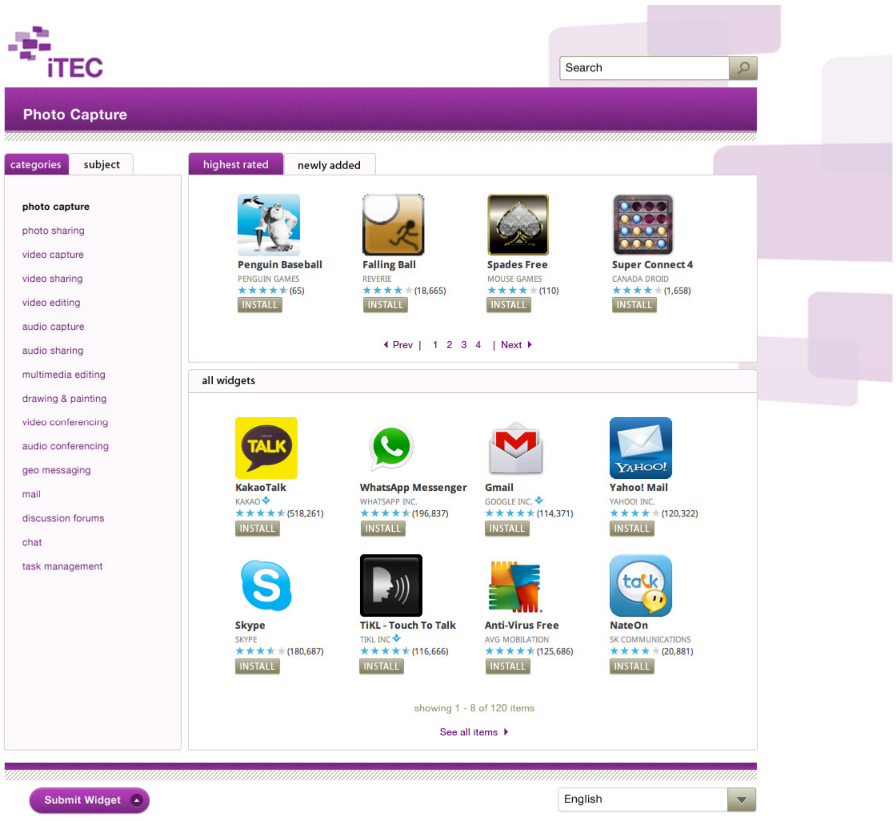


Figure 10: Wookie AppStore front-end

This screen shot shows how a user can simply view all of the widgets in a selected category, again ordered by affordance. In this view it is possible to see how many widgets there are under a selected category.

It is hoped that with a common interface and simple searching tools, teachers will be able to experiment with the deployment of different types of widgets. For searching tools based on critieria that teachers prescribe, we have we have implemented a 'Widget discovery service'.

2.3 AppStore Widget Discovery Service

Central to the usability of the iTEC AppStore is the ability of teachers to discover new tools based on search criteria. For this, Work Package 8 has implemented a 'widget discovery service'. This is a backend search engine for widgets that are stored in Wookie. It currently allows widgets to be searched upon the meta-data stored in Wookie.

The search engine runs as a separate service that sits alongside Wookie. The principle behind this separation is to allow the discovery service to be flexible and extensible. For instance, there may be a number of running instances of Wookie with interesting widgets installed in different locations. The discovery service could be configured to search all or a number of these instances. As far as the iTEC project is concerned it could also be configured to work with the iTEC registry service when the service becomes available.

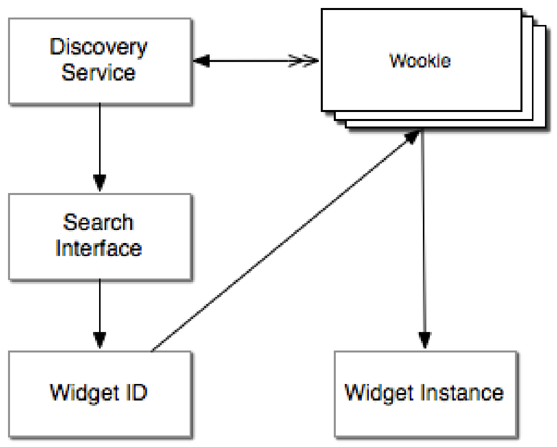


Figure 11: Wookie discovery service architecture

Figure 12 shows how information flows between the discovery service an interface and Wookie. The user would see the search interface as a text box which would be embedded in the plug-in and be presented with a list of results from which a widget could be chosen. Choosing the widget would send a request back to Wookie with the Widget ID. Wookie would respond by getting an instance or creating an instance of the widget and the instance information back to the plug-in so it can be displayed.

The discovery service makes use of Apache Solr/Lucene. The search language is Lucene and the search engine is Solr. Currently this is run with an embedded server (Jetty). But it can also be configured to run as a servlet under Tomcat or similar.

The search engine is written as a cluster of search cores, which communicate with Wookie via a REST interface. The list of widgets is returned in ATOM format, categories in JSON.

As the discovery service is standalone it is not yet embedded within the plug-in framework so therefore plug-in developers need to build the technology in in the plug-in gallery view. This starts to look like a store interface – however – each store would be written separately making use of disparate components and embedding a lot of the store logic within the plug-in. This makes less sense than having a more coordinated store interface, which contains the features that are required and is callable from different plug-ins.

The discovery service is available at:

http://iecbolton.jira.com/svn/ITEC/widget_discovery_service/trunk

The service can be downloaded and run alongside Wookie, and the instructions on how to set it up are included with the code.

The designs for the iTEC Application Store presented in Section 2.2.1 will be implemented using this new architecture in the coming year. This will enable the effectiveness of the discovery service to be evaluated, and in the light of this further development work will be carried out on the discovery service.

2.3.1 Management of Server and Installation of Widgets

In Year 1, the Management of the Wookie widget server, including the upload of new widgets, is restricted to those users with the appropriate permissions to administer the Wookie server, or have administrator rights to the hosting machine.

Restrictive though these user rights currently are, they have allowed us to extensively test the mechanisms for uploading, deleting and updating widgets on the system.

Currently, there are three methods whereby new widgets may be added to the system:

Widget upload method 1

By logging-on to the Wookie server, an administrator can upload a widget or an OpenSocial gadget to place them into the AppStore. As administrators, they are presented with a range of options (shown in Figure 13) below:

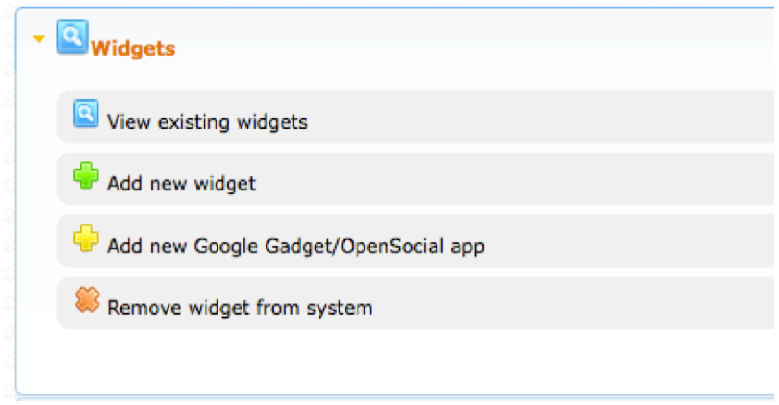

Widgets
View existing widgets
Add new widget
Add new Google Gadget/OpenSocial app
Remove widget from system

Figure 12: Wookie widget management interface

Currently, the process for adding a W3C widget is different from addition a Google Gadget. On clicking ‘Add new Widget’ a dialogue box appears prompting the user to upload a W3C widget file (.wgt). On clicking ‘Publish’, the .wgt file is processed and entered into the Wookie system.

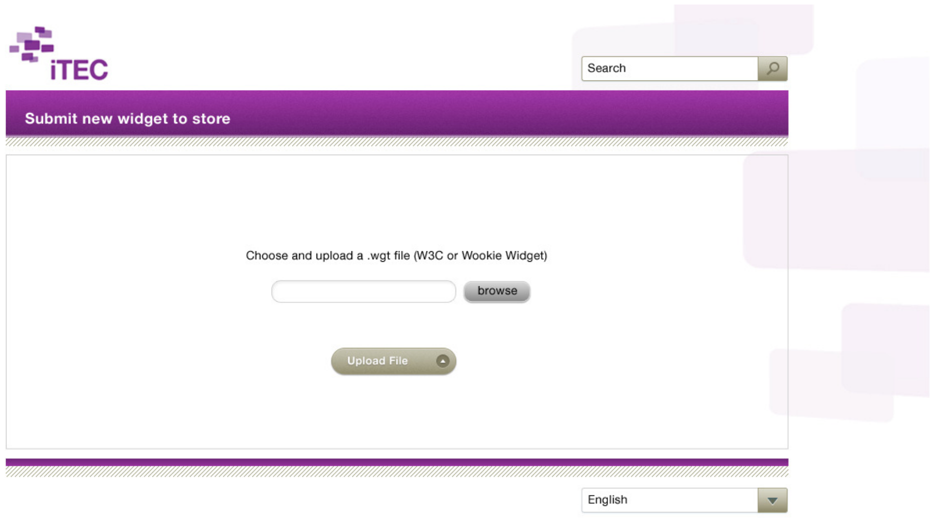

iTEC
Search
Submit new widget to store
Choose and upload a .wgt file (W3C or Wookie Widget)
browse
Upload File
English

Figure 13: Wookie widget upload page

Clicking ‘Add new Google Gadget/OpenSocial app’ prompts the user for the URL where the particular gadget is located, whereupon a similar publication process follows.

The system administrator who has administration access to the Wookie server through a web browser would primarily use this method. The admin usually accesses this functionality through the url:

http://your.wookie.host.url/wookie

Widget Upload method 2

For administrators with direct access to the Wookie installation directories on the server itself, a widget can be added directly to the 'hot deploy' folder in the Wookie installation on the server. In this way multiple widgets can be added at once. This is a method that would typically be used by server administrators for added widgets, maybe en-mass. It can be accessed with the following path

/path/to/appserver/webapps/wookie/deploy

Widget Upload method 3

The final method is primarily for widget developers who also have file access to the server installation. Scripts are included with Wookie for creating and deploying new widgets using Ant. Having launched a terminal application the developer needs to navigate to the Wookie installation widgets folder. The following ant scripts can be used:

- ant seed-widget
- ant deploy-widget

In each case the short name of the new widget is requested.

2.4 Application Store and Connector Framework integration

Figure 3 shows the architecture of the Wookie Connector Framework as it relates to the user experience of choosing Widgets that has been implemented in year 1. he diagram shows that the interface mechanism for Wookie is in the plug-in which makes possible the interoperability between the connector framework API and the host application (i.e. Moodle, Liferay, etc)

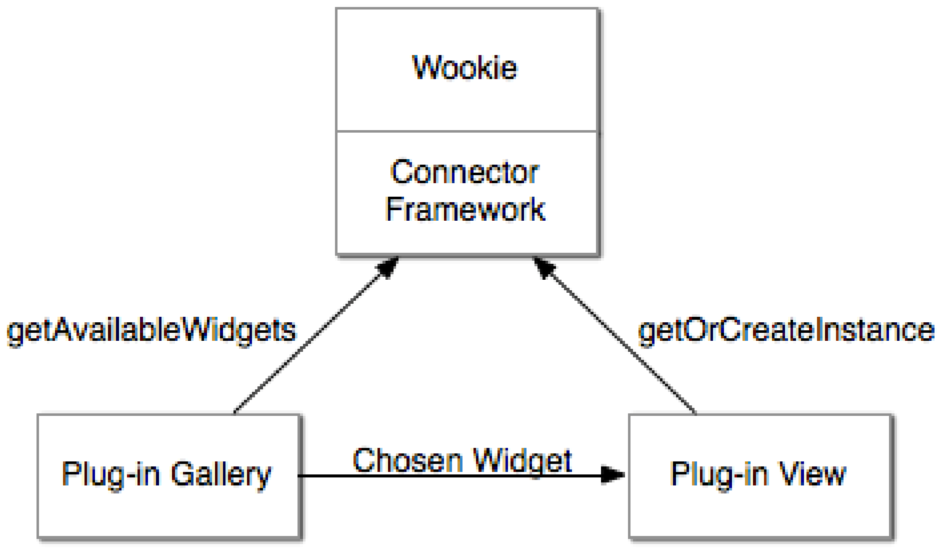


Figure 14: Wookie Connector Framework Architecture

However, a potential problem with this is that there is no real common user interface for the process of using a plug-in to instantiate a widget. Each shell environment could potentially have a totally different user interfaces for selecting a widget. Whilst this may or may not be desirable from a flexibility point of view it does make maintenance of the plug-ins and connector framework increasingly difficult. As Wookie grows as a service with various features, capabilities and APIs being added or deprecated as time goes by each piece of software, which makes use of Wookie also needs to be maintained.

Creating a store interface also affords the opportunity to create a reference implementation for consuming Wookie and search services. The implementation will have a logical flow as shown in Figure 4.

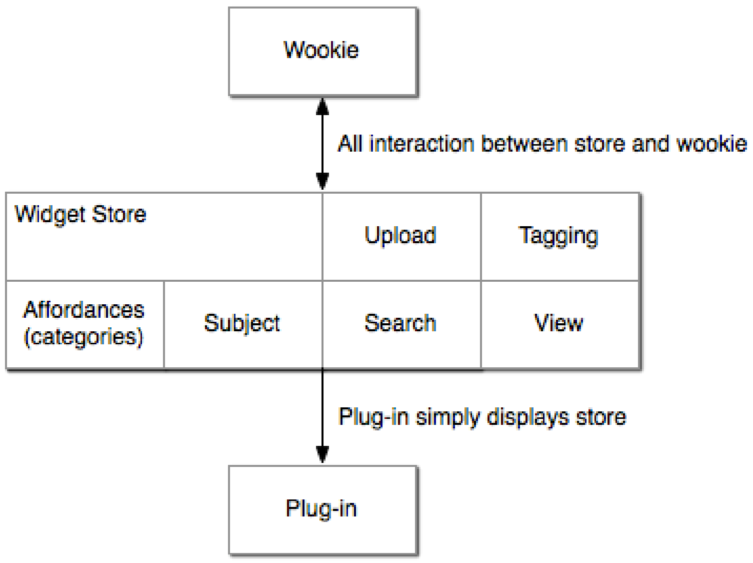


Figure 15: Revised Connector Framework and AppStore integration

This takes much of the weight of choosing a widget out of the plug-in and places it in the store. Each plug-in can take the store and simply display it in a way that is appropriate for the host platform.

2.4.1 Common authentication across shells

The issue of choosing a common authentication scheme across the iTEC project is something, which is being addressed during the second year of the project. The most likely candidate for this at the moment is oAuth (http://oauth.net). Whatever solution is adopted by the project will be adopted by the store. Again, having a single store rather than multiple connector-framework based plug-ins will make the transition to the adopted solution far easier to implement.

The information model required by the store is has five entities. Two of these entities – Widget (tool) and Affordance - map to models, which will eventually be maintained by the iTEC Back-end Registry currently under development in WP9.

The entities:

- Widget: externally to a Wookie with a widget id and contains, upload date, description and subject.
- Review: links to widget and user and contains a description and a rating.
- Affordance: links to widget and has a name or affordance title and a rating
- User: links externally to Wookie and contains an email.
- Event Log: links to nothing contains an id, time, description and title.

2.5 Enhanced management of Widgets in iTEC

All of these methods of widget inclusion are based upon the user having some element of control over the widget server itself. iTEC, however, has more flexible requirements for the installation of new widgets. As part of integrating the widgets more fully with shells the user of the shell (perhaps a teacher or an education coordinator) will need to be able to upload new widgets through the shell technology itself. So if Moodle or Liferay is being used as the shell environments, for instance, there needs to be the ability to upload widgets directly from the shell or even from within an administration widget, rather than having to be given admin access to the central Wookie repository.

The iTEC Application Store will give the user the ability to search and browse widgets on a richer set of descriptions, and also to upload new widgets and tag them with meta data required for the searching and browsing. This involves two aspects of work. Firstly, the underlying functionality of the server needs to be separated out and implemented as a Widget Discovery Service. Secondly the iTEC Application Store needs to be designed and implemented.

Work has been commenced on both these strands of work, with a working prototype and a design. for the application store. Delivery of the iTEC Application store is scheduled for month 24 of the project (September 2012).

2.6 Further development of the AppStore

The iTEC store will be an interface where widgets can be searched, installed, uploaded, tagged with categories and affordances and installed. It will be embeddable in shell applications in much the same way as the current connector framework system using native plug-ins in the target shells.

Currently each plug-in builds an interface that allows the administrator to choose a widget. This interface is simply a list or grid of widgets installed in a running instance of Wookie. This is fine if there aren't too many widgets. However, if the number of widgets goes above, say, twenty-five, then it becomes much harder to use. In the iTEC project there is a request to have as many as three hundred widgets installed to give the teachers a wide choice of tools and resources. Clearly the gallery interface as it currently stands would be impossibly unwieldy, even if pagination of the gallery were to be implemented. The store will address this issue by providing an interface that groups together widgets by categories and supplies a search engine utilizing all the widgets meta-data and extended meta-data.

There is also a requirement to allow new widgets to be installed via the shell applications rather than through Wookie's administrative interface. Currently this could be done by adding that capability to each plug-in or by developing a javascript interface for uploading and then creating a widget to do this job. A more elegant solution is to build this capability into the store. This also allows a workflow of widget inclusion to be developed. If we want teachers and maybe students to have the capability of upload and tag widgets either that they have created or that they have found then some kind of approval mechanism should be in place. The workflow envisaged is described in Figure 15 below.

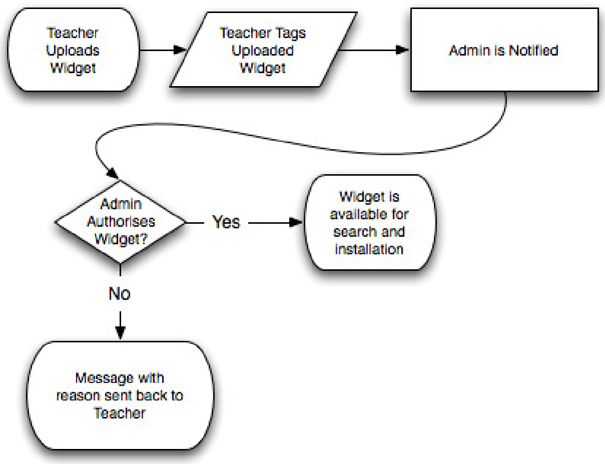

Teacher Uploads Widget
Teacher Tags Uploaded Widget
Admin is Notified
Admin Authorises Widget?
Yes
Widget is available for search and installation
No
Message with reason sent back to Teacher

Figure 16: Wookie widget upload workflow

As can be seen the flow there are two roles.

a. The teacher is actually anyone who might have an interest in uploading a widget. The type of people or roles that could do this is something that needs to be discussed within the project as it may have some impact upon pedagogic design.
b. The other role, here, is called Admin. Again, who this might be in the real world will require discussion. For the purposes of the project this person is best described as someone who has final control and ownership of the iTEC store contents.

2.7 Prototype implementation

Work has already begun on designing and implementing a prototype of the store. In the first instance the development is focusing on presenting the user with categories and subjects to break down the list of widgets available and also on integrating the existing widget discovery service.

2.8 Community Aspects of the AppStore

In preparation for the implementation of a community-driven AppStore, procedures and guidelines for the maintenance of a large-scale community online service have been provisionally detailed. These guidelines document policies relating to:

- User policies
- Data management and security
- Resource management
- Hardware
- IPR

It is envisaged that this initial specification of the running of the AppStore will evolve as practice within iTEC evolves, but the vision of these policies is that the iTEC AppStore should be open to all, and encourage the active engagement of the teaching community across Europe.

2.9 Valorisation and User testing of the iTEC AppStore

The key to the valorization of the iTEC AppStore is the engagement of teachers in the concept of searching and browsing new tools and instantiating them in their environments. A preliminary timescale for the testing of the concepts of the iTEC AppStore and its associated concepts of the implementation of scenarios with teachers has been established. The objectives of the test plan are:

- Identification of current teacher practice and current technologies
- Demonstration of integration of iTEC AppStore and Connector Framework within current environments
- Identification of Widgets of interest
- Identification of existing tools that might be ‘widgetised’
- Identification of barriers to adoption
- Targeting of key platforms favoured by teachers for the implementation of shell plugins.

Work Package 8 has worked particularly closely with the manufacturers of interactive whiteboards who already have detailed knowledge about the technological practices of teachers in the classroom. The approach to doing this is to start with an evaluation and testing of those components already established in the classroom environment which most closely resemble W3C widgets in their operation. Thus, a test plan has been formulated with data from the Interactive Whiteboard providers drawing on initial experiences of using W3C widgets and other interactive components that are readily available within the environment of existing Interactive Whiteboards. This data can then be used as a foundation to guide the implementation of iTEC widgets within the classroom in a realistic way which fits existing practices. The creation of a Wookie Widget plugin for the SMART Interactive Whiteboard is a key component in being able to move this further forwards.

The test plan is intended to feed into the processes of population of widgets so that those tools which are seen to be valuable by teachers (as well as important to particular scenarios) are prioritized in development within the project.

3. COMPONENTS OF THE ITEC WIDGET REPOSITORY FOR SCENARIO INSTANTIATION

In developing the Wookie Server as a connector framework which addresses the interoperability requirements of the project, and in creating an educational AppStore around it, Work Package 8 has sought to address the barriers to practice and to nudge teachers into new practices with widgets. However, iTEC as a whole is concerned with sustainable transformations of practice organised around the principle of educational scenarios. To support this, a range of technical developments within the project are brought together to describe and store scenarios (including details of activities, tools, people, events) with the aim of creating tools whereby teachers can instantiate and run effective scenarios with new technologies in their classrooms.

The instantiation of a scenario is therefore a different action performed by a teacher to the instantiation of individual widgets. For this the project requires that widgets and other tools are stored in a way where they be accessed and instantiated collectively based on scenario descriptions. In effect, this means that an iTEC 'repository' of widgets described in ways that meet the requirements of scenarios is also required. This work is closely related to the iTEC Application Store, but it is conceptually and technically distinct.

Work Package 8 has developed ways in which this problem of repository storage can be addressed, and widgets and other tools made available to the scenario instantiation process.

This work has proceeded by addressing three principle issues that relate to the iTEC repository:

1. The description of widgets in the repository in a way which can be related to the functionality required by scenarios
2. The provisioning of sufficient described widgets and other tools that meet the scenario requirements
3. The facilitation of means whereby teachers might create their own widgets and describe them to meet scenario requirements as necessary

In this work, the principle challenge has been found in 1) above (the description of widgets). This is because:

a) The nature of online tools is such that general descriptions of functionality mean that insufficient distinctions can be made between different tools. For example a forum and an email list have much the same function of presenting threaded discussions.
b) The selection of one tool over another (when they have the same functionality) depends on factors beyond the functionality of the tool (personal preference, custom and practice, institutional policies, technical barriers, etc).

This problem has been well recognised in previous work – notably in the EU-funded iCAMP project - and Work Package 8 has sought to find a solution to the description of tools by building on that work to create a solution that meets the specific requirements for iTEC.

3.1 Affordances, iCAMP and the description of tools

The solution to the problem of describing tools in a way which meets the requirements of specify the functional requirements of ‘what needs to be done’ in an activity without specifically identifying individual tools has been to code the descriptions of the various ‘things that need to be done’.

This is the approach adopted in the EU-funded iCAMP project (see http://www.icamp.eu) and the iCAMP tools have formed the principal inspiration behind the approach to tool description adopted in iCAMP.

iTEC has two specific requirements for a description framework for tools:

1. That tools recommended for instantiation within a particular scenario will work in the technical setting of a school. In other words, a match can be made between the technical capabilities of the school and the technical requirements of the tool.
2. That recommended tools the requisite functionality to perform the activities identified in a scenario.

In line with the approach adopted by iCAMP, the technical description of tools (their operating environments, language, interoperability capabilities/requirements, etc) has been separated from a description of what they do. We have followed the iCAMP approach of ‘Soft Ontology’ (see http://www.icamp.eu/wp-content/uploads/2007/05/d22 icamp building-blocks.pdf) to identify the things that are required to be done. This allows for the creation of an over-arching architecture which relates scenario description through to the instantiation of tools in technical settings:

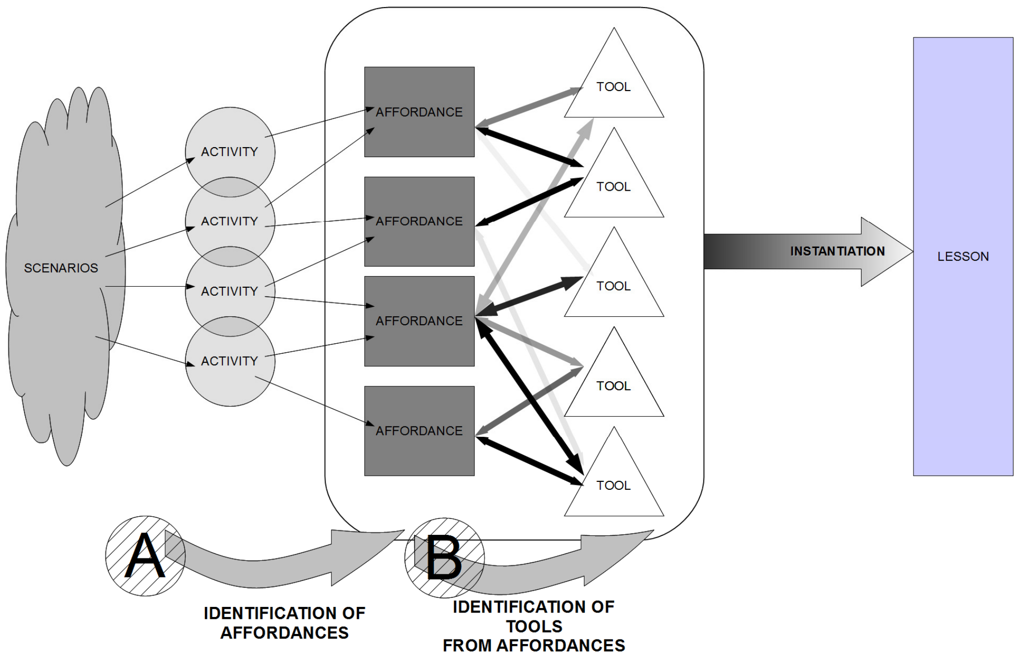


Figure 17: Scenario description to Affordances and Tools

Figure 17 illustrates how scenarios lead to 'activity descriptions', from which in turn, affordances can be identified (i.e. "in order to do this activity, we need to be able to…"). The affordances identified can then be related to particular tools according to a numerical scale indicating the degree to which a tool meets an affordance. This 'scale' is the principle mechanism by which the 'Soft Ontology' in iCAMP and iTEC is realized.

From the attachment of tools to affordances via the scalar linkages, recommendations can be made for the appropriate tools to realize the functionality specified in individual activities in the scenario, whilst also taking into account the technical capabilities of each individual technical setting within a school.

This descriptive work on tools essentially falls between the roles of a number of work packages in iTEC. Work Package 8 has identified a method of description because of the closeness of its work on maintaining a repository of 'tools'. Needless to say, this descriptive work has been done in close partnership with those other Work Packages whose ultimate responsibility it is to make recommendations for tools in technical settings. Thus, the descriptions of affordances and the descriptions of tools form part of the overall descriptive structure of the iTEC registry (Work Package 9), whilst the inferences of appropriate tools in appropriate contexts in the light of particular scenarios will be performed by the Scenario Development Environment (Work Package 10).

An example of how the description of affordances and the description of tools can be mapped-out is shown on Page 42, Section 3.3 "Relating Widgets to Scenarios: some examples".

3.2 Plan for Development for Searching and Tagging of Widgets

Initially, the tagging of widget affordance data will be implemented within the iTEC Application Store. This will involve a similar interface to that provided by the iCAMP software, where sliders are used against identified 'affordance' description for each tools, and each slider is adjusted to indicate the extent to which a tool demonstrates that affordance. In the Application Store, on uploading a new widget, the screen below (Figure 18) will appear to allow each widget to be tagged with an appropriate list of affordances.

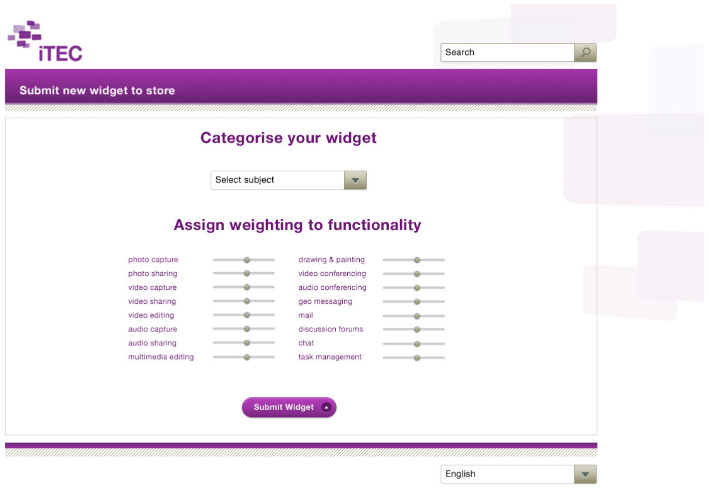


Figure 18: Wookie tagging affordances

This affordance data can then be stored in a way where the widgets are categorized for use within the AppStore environment, but also available for use by the scenario development environment.

3.3 Relating widgets to scenarios: some examples

3.3.1 A breath of Fresh air Scenario

In this scenario two teachers decide to organize students in groups, so that they can work collaboratively. One of the students (Carmen) is asked to go outside and take pictures of where ladybirds live.

> *"After gathering a series of photos Carmen comes back to class with her group and they share their data and findings with each other."*

In terms of "sharing photos", one would be inclined to think of an image sharing service such as Flickr or Picasa Web album, as the usual type of service to handle such a requirement. The affordances of being able to 'take a picture' would then be mapped onto the different technical capabilities of the various services and tools that might be invoked. However, it should be noted that in order to use such services, users would have to provide authentication credentials to enable logging into the service, uploading of photos, etc. With this in mind (and also to make the process much easier) a lighter image hosting web service was devised & created. This allows users to easily take photos & share them using a Wookie widget, without having to go through logging into Flickr or another service. It should also be noted that Widgets/Gadgets that allow a user to browse services such as Flickr already exist in other widget/gadget implementations which could be leveraged/imported onto Wookie. Later in the same scenario the following is mentioned...

> *"They get some specific support from Ms Rossi on how to use a software package to draw conclusions from the group's numerical data."*

This suggests affordances related to the ability to enter data in a spreadsheet (indeed, further down in the scenario, spreadsheet software is mentioned). With this in mind an idea to create a shared spreadsheet widget was devised. On entry into the iTEC AppStore, the affordances relating to the scenario can be matched with the affordances of the technology. However, the capabilities of the spreadsheet widget go beyond entering data: multiple users could enter data into the spreadsheet and one users view of the cell information would update automatically to reflect the changes made by another user. As a result, there are communicative affordances relating to the concepts of group working, and these too can be configured using the affordance idea (for example, the spreadsheet widget also has some affordance for asynchronous communication). This would reinforce the scenario where it mentions that this is a group activity. Why not use google docs? For the same reason as earlier mentioned when discussing the creation of an image sharing service. This way we avoid the problems of students having to have created google accounts before they can access data and other authorization issues.

This widget is partially completed and currently still under development. This scenario also mentions file editing software...

> *"They work together using laptops and a web tool to create a short digital film explaining what they found"*

Some ideas around this could involve the development of new widgets that could support online video editing. On line software examples include http://www.youtube.com/editor & http://jaycut.com/. With the former example, a 'You-tube videos' widget could then be used to view the resulting videos that have been created.

3.3.2 Widget requirements from other scenarios

The following scenarios mention a number of tools which could be implemented as widgets (or already exist as implemented widgets/gadgets). This list was created by examining selected scenarios and looking for functionality which could be made possible by use of widgets.

Beam in the Expert

- Social Networking
  - Facebook
  - Twitter
- Instant Messaging
- Forums
- Video conferencing
- Document sharing (either using an existing service, i.e. dropbox or implemented in a similar fashion to the above 'image sharing service')

Biblo High Tech

- Social Networking
  - Facebook
  - Twitter
- Video Uploading

Supported through Change

- Instant Messaging
- Video conferencing

Figure 19 shows the Wookie widget server integrated with the appropriate interfaces for the description of scenarios. Wookie is represented at D) performing two roles:

a. As a connector framework which feeds widgets to A) which are tagged with affordances at B)
b. As an AppStore which provides functionality for users to be able to rate widgets at C).

The meta-data associated with a widget includes

- Name
- Icon
- User rating
- Categories
- Subject
- File Size
- Version
- Number of downloads
- User comments

This basic data can then be added to by users of the AppStore by contributing their own comments. The AppStore provides functionality to install the widget either on a shell or device.

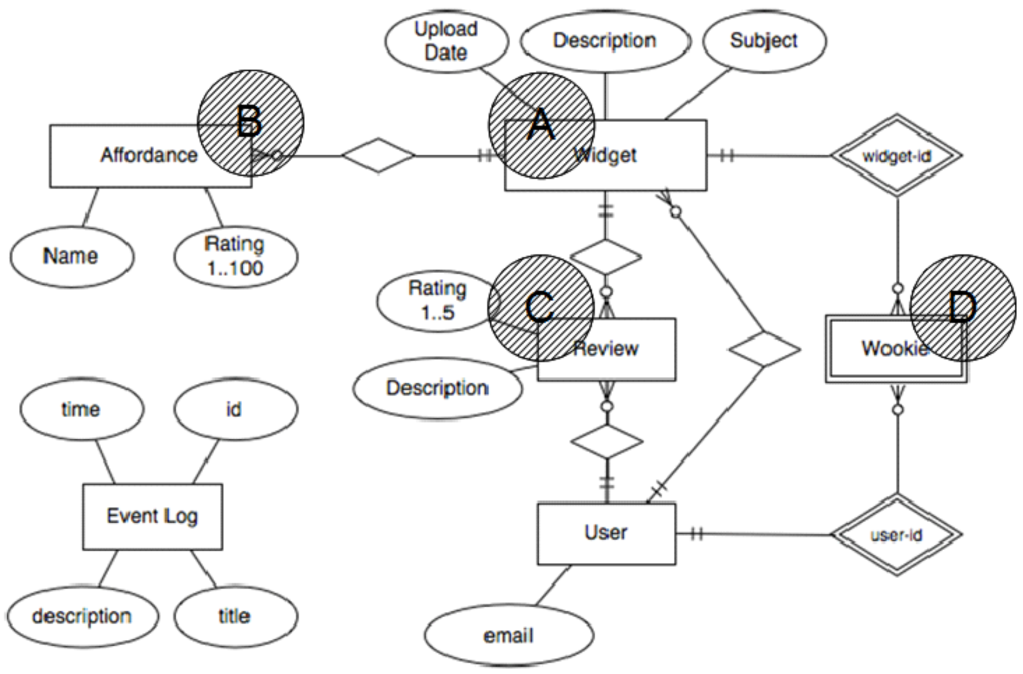


Figure 19: Application Store relation to Affordances and User ratings

Tools already in existence may be identified and brought within the iTEC environment (Wookie) and their basic affordances described. This is indicated at (B). This would allow those tools to be considered as potential ways of realising the toolkits of individual scenarios.

Tool not currently in existence may be identified by teachers who recognise the need to realise particular affordances with either existing or new scenarios. The means by which existing tools with existing affordances can be adapted by teachers and placed within the iTEC environment is the subject of the Widget creation environments discussed further below in this document. This is indicated in the Figure 19 at (C).

3.4 Widget Provisioning and scenarios

The description of scenarios specifies the tool requirements to support their activities in the form of affordances. Somehow, the Wookie Server needs to contain the appropriate tools with appropriate affordances that meet the requirements of each scenario. The population of widgets within the Wookie Server can occur in three ways:

1. New widgets will be created to meet the affordances of a scenario
2. Existing widgets will be tagged to show how they meet the affordances of a scenario
3. Existing widgets will be discovered on the web and imported and tagged in Wookie to meet the requirement of a scenario
4. Tools will be provided to facilitate the creation of widgets by teachers to meet the affordances of a scenario.

Once the infrastructure for delivery and management of widgets is in place this will need to be populated with widgets for use in scenarios. There are many widgets available, but it is also expected that additional widgets will be created by iTEC where there is a requirement generated by a scenario which cannot be met by an existing widget.

Apache Wookie by default contains a number of widgets, including Instant Messaging and vote/poll type widgets amongst others. However, there are many other existing widgets to be found around the internet. Not only does Apache Wookie allow for any W3C compliant widget to be imported into its bank of widgets, it also allows the use of Google Gadgets to shown via it's Apache Shindig integration.

In the coming months the project will survey the available widgets for use in pilots. Where none is available, it will be necessary to develop something tailored towards iTEC . As is mentioned some work has already been carried out in this direction, as it seems very likely that some kind of spreadsheet functionality will be required.

An early version of the spreadsheet widget can be found here:

http://iecbolton.jira.com/svn/ITEC/widgets/trunk/spreadsheet_widget

Work is required to format the widget contents and to link the save functionality to Wookie's persistence mechanism. Here is how it looks currently:

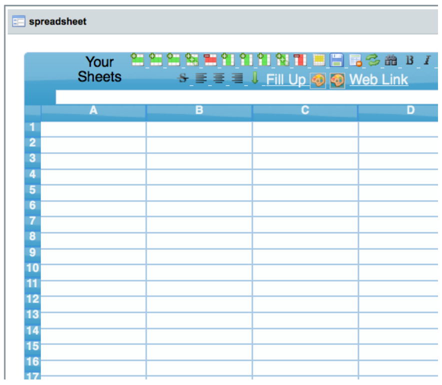


Figure 20: Wookie spreadsheet widget

Currently the plan is to have this widget available in month 24.

3.4.1 The harvesting of existing W3C Widgets into Wookie

Having established clear criteria for the labeling and tagging of widgets, a process of populating the iTEC AppStore with widgets that might be of interest to teachers has been engaged with. While it is clear that there are many widgets to be found, some will be more relevant to iTEC than others.

Places where W3C widgets can be found include:

http://widgets.opera.com/ (W3C Opera Widgets compatible with Wookie: > 1000)
http://www.google.co.uk/ig/directory (Location for Google Gadgets, compatible with Wookie: > 1000)

http://shop.vodafone360.com/shop/ (W3C Widgets compatible with Wookie: >1000)

http://arc.tees.ac.uk/wide/Resources.ashx (Wookie Widgets from the WIDE Project: aprox 25)

There are a number of references to social networking tools in the cycle one scenarios. Several google gadgets can be found offering Facebook, Flickr and Twitter, interaction for example, and it may be possible to harness some of these widgets for iTEC uses. Viewing and manipulating videos is mentioned in at least two of the scenarios, so widgets that allow this, such as a You-Tube widget, would be very relevant. Video Conferencing is another tool which is mentioned in the use cases.

3.5 The development of Widget Creation tools

Widget creation tools are in development which will facilitate the generation of new widgets based on templates of existing widgets. We propose to adapt our widget creation tools from existing tools that base widget creation around a template. This will afford the teacher the ability of being able to fill the gap of that tool by adapting existing tools.

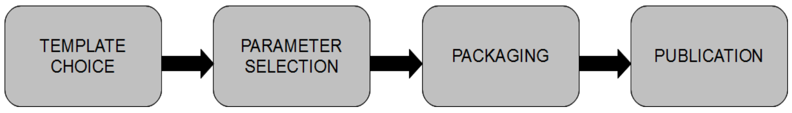


Figure 21: Processes for Widget Creation tools

Since the W3C widget is simply a web page, simple experiments in Widget Creation services have already been tried. For example, a 'wrapping' of internet pages as widgets can be useful as a way of sharing and embedding existing applications within the environment of the iTEC shell. Similarly, existing standalone web applications, whether in the form of simulations as Java applets, or other games can all be wrapped up and deployed within an iTEC shell.

Our intention is to continue this development work over the course of the project, producing Widget templates and identifying new patterns which are effective for teachers to customise and develop their own widegets. In this endeavour, we will be guided by and contribute to other work which is based around the process of creating widget makers, including the JISC WidGAT project at the University of Teeside.

3.6 iTEC Repository Testing Plan

The testing of the iTEC repository is related to, but distinct from the testing of the AppStore. On the one hand, the testing of the repository involves:

- Ensuring that widget descriptions and tagging lead to effective inferences
- Ensuring that scenarios are realized with tools that work within the context of the scenario
- Ensuring that appropriate tools are instantiated in approach settings

Unlike the context of technical testing, pedagogical testing exists within the context of individual scenarios, the classroom environment, the learning contexts of children and teachers, and the readiness to engage with iTEC technologies and the learning outcomes of children and teachers. Results from this testing will inform the appropriateness of particular tools to individual scenarios. It is likely that at this stage, deeper evaluation of the realistic ambitions of individual scenarios will also be exposed.

In preliminary results using Promethean Smartboard technology, teachers reactions to the initial scenarios and the currently available tools through the Interactive Whiteboard (and other tools) have provided valuable indicators as the appropriate direction for development and deployment of widgets within the context of those scenarios.

The shells that are currently being targeted to support the pedagogic groups.

Curent technical testing work is involving the following target shells:

1. Moodle
2. LifeRay
3. Interactive Whiteboards (SMART)
4. DotLRN

Further integration testing involves the redeploying of instantiated widgets across a range of different technologies including:

- Blogs
- Wikis
- Mobile phone OS (Android)

The criteria for this testing phase involve identifying the ways in which the iTEC interventions help teachers to present their ideas to learners through the provisioning of tools and resources, and the ways in which learners may present their ideas to each other and to their teachers.

The broad aim of iTEC is to increase the variety of ways in which these things can be done. For example, IWB technology presents a simple means by which ideas may be expressed. Introducing iTEC Widgets potentially increases the ways ideas can be expressed through the use of a richer toolset.

Potentially, this increases the range of possible activities, but does it also increase the ways in which activities can be coordinated?

It is envisaged that through this testing regime, the functional needs of addition widgets may be identified.

4. SPECIFICATIONS AND STANDARDISATION

4.1 The standards landscape for WP8

Recently there has been a surge of interest in web standards as opposed to device specific or language specific standards. Several organizations and standards bodies have been working in this space for some time and consequently a rich set of specifications are emerging – some of which are competing and some of which are complimentary to one another. We have W3C widgets as a standard which is ongoing and developing, WAC/JIL Widgets, Chrome widgets but their adoption of W3C as their main standard, Mozilla Open Web Apps, OpenSocial Gadgets all of which provide the same fundamental web standards but with key differences between them which makes interoperability hard.

The table on the following page provides an overview of some of the most relevant standards for WP8 work. For a higher resolution image please see:

http://scottbw.files.wordpress.com/2011/02/landscapev2.png

Most of the packaging formats are similar in that they are a zip archive containing a number of files some with content and functionality some with meta-data describing how they should be displayed, author's name, pointers to icons etc.

The W3C version has now been adopted as a common packaging format for a number of different app stacks such as the Wholesale Application Community and in the specifications for Opera's extensions and unite platform.

Recently the Wireless Applications Community adopted the W3C widget interface specification. This is of value to iTEC because it means that interoperability with Android widgets is now practicable and easy to achieve. This development is symptomatic of the upward trend in adoption of W3C widgets by vendors and organizations over the course of the past year.

However, social, user and data persistence functionalities are required by iTEC and we have to look elsewhere to develop these features. Currently the Google Wave API is being used by Wookie widgets and our own extensions with shared data. However, as the web app landscape takes shape other solutions may be adopted or added in as features such as XMPP for collaboration or similar. iTEC will contribute to this decision within the Apache Foundation.

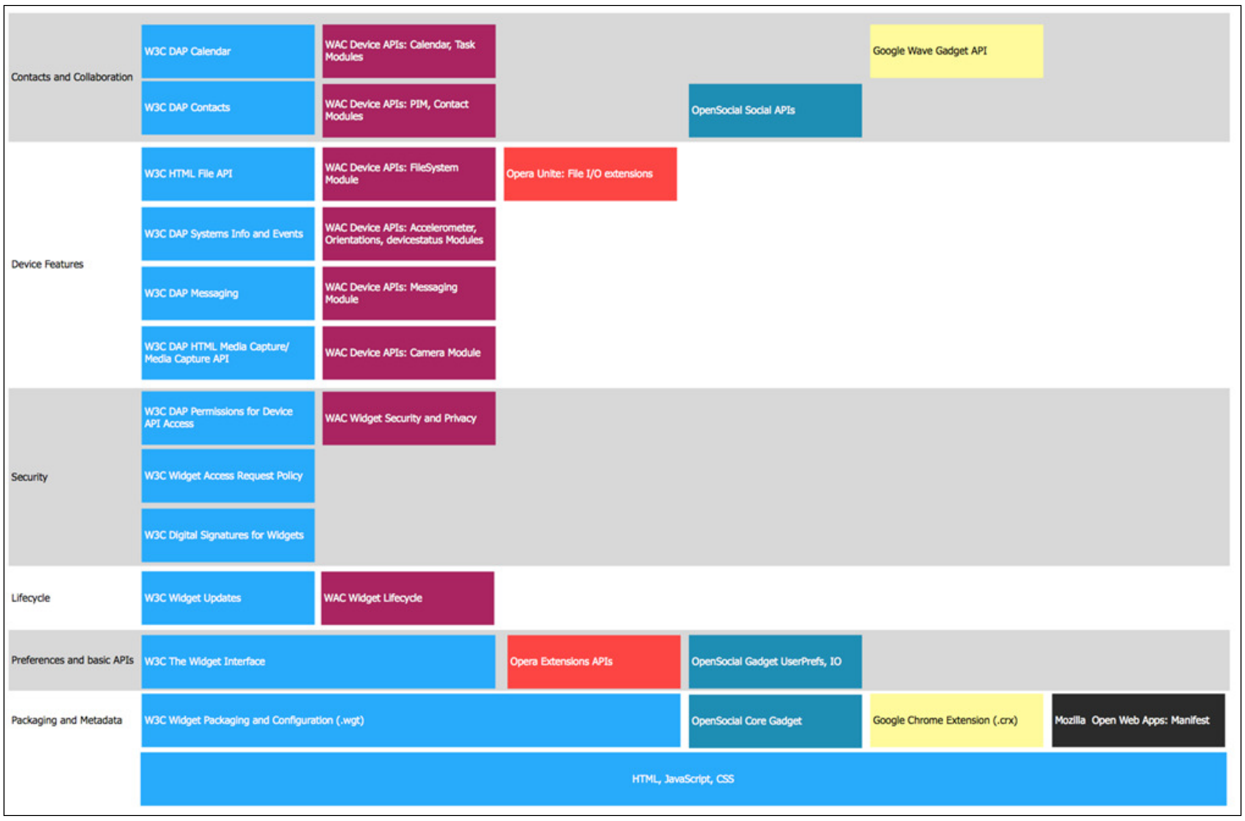

Contacts and Collaboration
W3C DAP Calendar
WAC Device APIs: Calendar, Task Modules
Google Wave Gadget API
W3C DAP Contacts
WAC Device APIs: PIM, Contact Modules
OpenSocial Social APIs
Device Features
W3C HTML File API
WAC Device APIs: FileSystem Module
Opera Unite: File I/O extensions
W3C DAP Systems Info and Events
WAC Device APIs: Accelerometer, Orientations, devicestatus Modules
W3C DAP Messaging
WAC Device APIs: Messaging Module
W3C DAP HTML Media Capture/ Media Capture API
WAC Device APIs: Camera Module
Security
W3C DAP Permissions for Device API Access
WAC Widget Security and Privacy
W3C Widget Access Request Policy
W3C Digital Signatures for Widgets
Lifecycle
W3C Widget Updates
WAC Widget Lifecycle
Preferences and basic APIs
W3C The Widget Interface
Opera Extensions APIs
OpenSocial Gadget UserPrefs, IO
Packaging and Metadata
W3C Widget Packaging and Configuration (.wgt)
OpenSocial Core Gadget
Google Chrome Extension (.crx)
Mozilla Open Web Apps: Manifest
HTML, JavaScript, CSS

Figure 22: Wookie and W3C related standards

4.2 Use and extension of specifications in iTEC

The widgets used by iTEC to implement this solution conform to the W3C widget specification. The W3C's Widget Packaging and XML document (http://www.w3.org/TR/widgets/) describes widgets as *fully fledged client side applications which are unlike traditional user interface widgets such as buttons, toolbars and input boxes. These applications are authored in client side technologies such as HTML, Javascript and Flash*.

A major stream of the work carried out by WP8 focuses on contributions to the Apache Wookie server, which is a reference implementation of the W3C widget specification. Consequently the extensions to the functionality of W3C widgets which are implemented in the server become candidates for submission to W3C. In the current state of work this is true of the connector framework and connector plugins. As work progresses on iTEC it is anticipated that more extensions to the capabilities of W3C widgets will be required. In particular the persistence of data between sessions to enable file-sharing, chat and other community functionalities will be needed.

It is also anticipated within the Description of Work that it will be necessary to support message passing between widgets, so that they can interact. In this regard iTEC is collaborating closely with the Omelette project, in which the University of Bolton is also a partner. It is hoped that some of the solutions developed by Omelette will contribute to the work of iTEC, thereby demonstrating the value of standards based systems.

As the project matures the extensions to the W3C widget specification developed by iTEC will be gathered and submitted to W3C.

The standards based approach has been extended to plug-in development. The plug-in for Liferay has been has been written in Java as a Portlet using the JSR 286 (ref) specification. This means that it will work in any environment that supports JSR286.

5. CONCLUSIONS

Work Package 8 is facilitating greater interoperability between tools in learning platforms, and through so doing, is helping to prepare the technical ground for scenario-based pedagogies. Scenario-based teaching and learning requires a rich range of tools which are easily to-hand, work with one another, and work across different learning contexts and platforms.

The principal challenges for Work Package 8 involves scaling-up widget-based services in ways which:

a. meet the direct needs of scenario-based activities
b. stimulate teachers to explore a rich range of tools
c. stimulate processes of scenario creation with new technical possibilities
d. make the creation of new widgets accessible to teachers
e. describe available tools in ways where effective and appropriate technical recommendations can be made
f. deploy widget technologies to fully exploit the technologies teachers already have in the classroom

The background to this work remains challenging: changing the practice of teachers is difficult. Yet the work on Mashup connectors within iTEC presents an opportunity for doing something that is innovative, that can integrate seamlessly with existing toolkits and existing practices, and whose potential for transforming the organisation of learning around interoperable toolkits is an exciting prospect.

APPENDIX I

Connector Framework Widget Specification

Wookie supports and extends the W3C specification for widgets. It understands how to unpack and deliver these to a browser or mobile environment. For more information about this specification please see (link to W3C widget spec).

Wookie does extend the specification for Wookie widgets to support the concept of the user and the concept of shared data.

■How does it do this?

Wookie takes a standard widget and injects extra features into the main file of the widget. These features can then be used within the html of the widget main page using javascript. So a the standard html header of a widget may look like this.

```
html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Default Chat Widget</title>
  <script type='text/javascript' src='chat.js'> </script>
  <script type='text/javascript' src='smileys.js'> </script>
  <script type='text/javascript' src='jquery.js'> </script>
  <link rel="stylesheet" href="chat.css" type="text/css"/>
</head>
```

After Wookie has parsed and delivered the widget the header looks like this.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  <meta http-equiv="PRAGMA" content="NO-CACHE" />
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Default Chat Widget</title>
  <link rel="stylesheet" href="chat.css" type="text/css" />
  <script type="text/javascript" src="/wookie/dwr/util.js"></script>
  <script type="text/javascript" src="/wookie/dwr/engine.js"></script>
  <script type="text/javascript" src="/wookie/dwr/interface/WidgetImpl.js"></script>
  <script type="text/javascript" src="/wookie/shared/js/wookie-wrapper.js"></script>
  <script type="text/javascript" src="/wookie/dwr/interface/WaveImpl.js"></script>
  <script type="text/javascript" src="/wookie/shared/js/wave.js"></script>
  <script type="text/javascript" src="chat.js"> </script>
  <script type="text/javascript" src="smileys.js"> </script>
  <script type="text/javascript" src="jquery.js"> </script>
</head>
```

The extension specification for a Wookie widget is supported through the use of direct web remoting (DWR) which allows javascript calls within the widget to be relayed back to Wookie. The Wookie wrapper javascript file makes use of DWR and adds the following functionality.

Function	Description
preferenceForKey	Gets a preference given the key. This is a W3C required function which either sets the preference locally or remotely depending upon the capability of the host environment.
setPreferenceForKey	Sets a preference given the key. Again this is a W3C required function.
sharedDataForKey	Gets data that can be shared between widgets. This is Wookie extension to the specification allowing data to be persisted on the server side and shared between users of a widget.
setSharedDataForKey	Sets some shared data. This can be any kind of data. It is persisted on the server side in a large binary field.

In addition to extending the data saving and sharing capabilities of the widget the extension specification also supports Google Wave (link to wave site).Whilst Google is not supporting wave itself anymore, the Apache foundation have an open source implementation, which should continue it as a server side library. In the case of Wookie the API is implemented fully within Wookie itself and allows widgets to make use of real time collaboration between users. Ongoing development within Wookie may see it being deprecated and replaced with a newer more flexible technology should one appear that conforms with the Apache Open Source licensing requirements.

APPENDIX II

Connector Framework Plug-in Specifications

Each plug-in delivers a consistent experience to the user whilst maintaining the interaction with the host shell.

On the technical side the plug-in has to retrieve the following information from the host environment or from preferences indicated by the administrator who configures the plug-in in the host environment.

The Host

This is a URL that points to the running instance of Wookie. The plug-in normally allows this to be set as a preference. It is in the standard URL format: http://wookie.site:portnumber/wookie

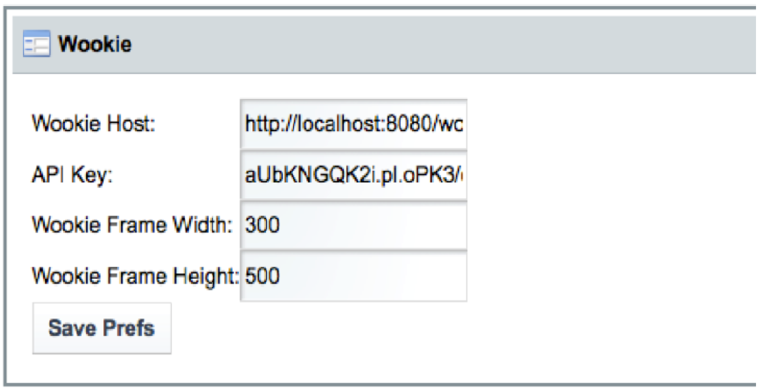
There is no need for the instance of Wookie to be running on the same machine as the shell environment but consideration should be given to the port and any security which the viewer of the site might have on the internet environment they are viewing from. For instance most institutions have all ports disabled for web viewing apart from port 80 so for the widgets to be viewable Wookie would need to be on port 80. As Wookie runs under an application server the default port setting might normally be port 8080.

The API Key

An *API Key* is used to access many of the features of the Wookie REST API. Each individual web application needs its own *API key*. *API Keys* are generated from Wookie's administration interface.

The plug-in allows this to be set as a preference. Wookie uses the API Key to identify the host application.

The following image shows the preferences for host and API Key that are enabled for the Liferay plug-in as an example of how this an be done.


Wookie

Wookie Host:	http://localhost:8080/wc
API Key:	aUbKNGQK2i.pl.oPK3/
Wookie Frame Width:	300
Wookie Frame Height:	500

The Shared Data Key

The *shared data key* is an arbitrary identifier that marks *widget instances* as being sibling instances that can share state information. It is up to the *plug-in* to determine this value; typically there is a persistent identifier available for whichever view is being used as the container for a widget. This is not normally a preference that the administrator of the plug-in can change.

Viewer

The *viewer* is the current user who is viewing a widget in the browser. Typically the host shell uses session information to determine who the current user is, and this is used to request a particular *widget instance*. It is up to the *plug-in* to determine how to identify the *viewer*; for example the user's real id is one possibility; another is an opaque hash-code using the id.

Process

A typical process undertaken in code by the plug-in is as follows.

1. Read preferences for API Key and Host
2. Find Shared Data Key
3. Find User ID
4. Use Host, API Key and Shared Data Key to create a WookieConnectorService
5. Set the user
6. Display a gallery of Widgets
7. Get or create a widget instance
8. Provide the instance URL to the plug-in view mode or host environment.

The above process is programming language independent. The same process is undertaken whether writing in java or PHP for instance. Support in the framework is currently there for the following programming languges:

- CSharp
- flash or flex
- java
- php
- python
- ruby

APPENDIX III

Summary of progress and plans

Widget Creation Tool Specification – Month 15

For month 15 we intend to create the functional and technical specification for a widget creation tool. This will be aimed at allowing teachers to create data collection widgets for use by students in support of iTEC scenarios and to develop new tools to monitor activity in the physical environment (e.g. GPS or RFID), or in the broader online world, to allow for scenarios that involve the use of physical artefacts, movement or online exploration.

We will spend some time identifying other widget creation tools provided by iTEC partners or other projects which can be made available for use in iTEC.

Repository of Widgets

Work will continue establishing which the tools (widgets) are required to support to support iTEC scenarios and Learning Activities.

However, it is recognized that having a rich set of widgets will be important for flexibility and choice with this in mind we have in place a plan to import 200+ widgets into Wookie server to be used in iTEC shells, tagged with the iTEC App store description framework including affordances. Widgets identified will be those with relevance to iTEC scenarios and learning activities identified in cycle 1 and 2.

A demonstration of available widgets and summary report will be produced.

Widget Creation Tool – Month 20

By month 20 a functional widget creation tool for use in the iTEC pilots will be available. It will allowing teachers to create data collection widgets for use by students in support of iTEC scenarios. Also new tools to monitor activity in the physical environment (e.g. GPS or RFID), or in the broader online world, will be developed to allow for scenarios that involve the use of physical artefacts, movement or online exploration.

Additional widget creation tools made available from other project and partners will be made available for pilots where appropriate.

App Store – Month 24

The Apps store will be further developed and implemented as described in the design document provided at the end of year one. This will include an initial draft set of user guidelines, policy guidelines, user security guidelines etc. The community server will be released for access by users. Early theories of adoption and use by communities will be described and explained.

A community approach will be designed for use and ongoing development, for example partners should be able to add new tools and technical developments. Procedures and protocols for the effective support and management of the Widget server will be developed encouraging a self-sustaining community-driven activity of updating, maintaining and sharing to support the ongoing use and development of the Widget Server.

APPENDIX IV

Wookie REST API

This is a draft specification for the Wookie REST API. This is the API invoked by web applications for actions such as requesting an instance of a Widget, getting a list of available Widgets, and for managing participants and properties for Widget instances.

API key

Note that except where noted, all methods require the requesting application to have a valid API key issued by the Wookie server administrator.

Widget Instances

See also https://issues.apache.org/jira/browse/WOOKIE-34
An instance is identified using a combination of the following parameters:

- *api_key*: The key issued to a particular application
- *shareddatakey*: The key generated by an application representing a specific context (e.g. a page, post, section, group or other identified context)
- *userid*: An identifier (typically a hash rather than a real user Id) issued by an application representing the current viewer of the widget instance
- *widgetid*: The URI of the widget this is an instance of (optional, see servicetype below)
- *servicetype*: Where an individual widget is not requested by URI as above, this parameter should contain the category of widget to be instantiated, e.g. "chat"
- *locale*: The preferred locale of the widget, expressed using a BCP47 Language Tag

Action	Request	Example Response	Description
	GET {wookie}/widgetinstances		Not supported.
Get or Create instance	POST {wookie}/widgetinstances {params:*instance_params*}	<widgetdata> <url>URL TO ACCESS WIDGET</url> <identifier>IH6rjs75tkb6I.pl.k0hUq7YdnFcjw.eq.</identifier> <title>Weather</title> <height>125</height> <width>125</width> <maximize>false</maximize> </widgetdata>	Either creates a new instance for the given parameters, or retrieves an already-created instance. If a new instance is successfully created, the response has a HTTP status code of 201; if an instance if successfully retrieved, a status code of 200 is returned.

	PUT {wookie}/widgetinstances {params:*instance_params*, action, [coneshareddatakey]}		Either stop, resume, or clone an instance, depending on the content of the *action* parameter. If the action is "clone", a shared data key for the clone must be provided using the "cloneshareddatakey" parameter.

Widgets

Action	Request	Example	Description
	GET {wookie}/widgets{?all=true, locale=*language_tag*}		Returns an XML representation of the set of available widgets. Note that this does not require an API key. If the "all=true" parameter is omitted, the list only contains the default widgets for defined service types. If a locale is specified, the returned information is localized, for example widget titles, descriptions, license information will be in the specified language where available.
	GET {wookie}/widgets/{service_name} {?locale=*language_tag*}		Returns an XML representation of the set of widgets that belong to the given *service_name*. For example, all widgets categorized as "weather". (See issue WOOKIE-10). If a locale is specified, the returned information is localized, for example widget titles, descriptions, license

			information will be in the specified language where available.
	GET {wookie}/widgets/{id} {?locale=*language_tag*}		Returns an XML representation of the widget with the specified *id*. Note that in the current release this is the actual database key; future releases should implement this using the widget URI as the *id*. If a locale is specified, the returned information is localized, for example widget titles, descriptions, license information will be in the specified language where available.

Services

Action	Request	Example	Description
	GET {wookie}/services {?locale=language_tag}		Returns an XML document containing all services and any widgets associated with the service category. If a locale is specified, the returned information is localized, for example widget titles, descriptions, license information will be in the specified language where available.
	GET {wookie}/services/{service_name} {?locale=*language_tag*}		Returns an XML representation of the service specified by *service_name* and all the widgets associated

			with it. If a locale is specified, the returned information is localized, for example widget titles, descriptions, license information will be in the specified language where available
	POST {wookie}/services/ {param:name}		Creates a new service with the name provided using the *name* parameter. If there is already a service with this name, a http 409 (conflict) error is returned. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.
	PUT {wookie}/services/{service_name} {param:name}		Renames the service specified by *service_name* with the new name given by the *name* parameter. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.
	DELETE {wookie}/services/{service_name}		Deletes the service specified by *service_name*. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.

Participants

A Participant consists of a participant_display_name, participant_id, and participant_thumbnail_url. Participants are always defined in relation to a specific Widget Instance; requests affecting one Participant have no effect on Participants associated with other Widget Instances.

Representation

Participants are represented in XML as a <participants> document, containing a <participant> element for each participant, with the attributes:

- id: the participant id (not to be confused with Wookie's internal id of a participant)
- display_name: the participant display name
- thumbnail_url: the url of the thumbnail image for the participant

Action	Request	Example	Description
	GET {wookie}/participants		Not supported.
	GET {wookie}/participants {params: *instance_params*}		Returns an XML representation of the Participants associated with the Widget instance specified by {instance params}.
	GET {wookie}/participants {params:id_key, api_key}		Returns an XML representation of the Participants associated with the Widget instance specified by {id_key}.
	POST {wookie}participants {params: *instance_params*, participant_id, participant_display_name, participant_thumbnail_url}		Adds a participant to the specified Widget Instance. If successful, a HTTP status code of 201 is returned. If there is already a participant that matches {participant params} for the instance, a HTTP status code of 200 is returned.

	DELETE {wookie}/participants {params: *instance_params*, participant_id}		Deletes the specified Participant from the specified Widget Instance.

The Properties API contains methods for both Preferences (properties affecting a single Widget Instance) and Shared Data (properties affecting sibling Widgets).

A property consists of a propertyname and propertyvalue.

Action	Request	Example	Description
	GET {wookie}/properties		Not supported.
	GET {wookie}/properties {params: *instance_params*, propertyname}		Returns the value of the specified property for the specified instance.
	POST {wookie}/properties {params: *instance_params*, propertyname, propertyvalue, [is_public=true]}		Sets a property for the specified instance. If is_public=true is set, the property set is a Shared Data entry; otherwise it is a Preference.
	PUT {wookie}/properties {params: *instance_params*, propertyname, propertyvalue}		Updates the value of the specified property of the specified Widget Instance.
	DELETE {wookie}/properties {params: *instance_params*, propertyname}		Deletes a property. This method returns a 404 status code if there is no matching property.

APPENDIX V

Wookie Admin REST API

This is a draft specification for the Wookie Admin REST API. This is the API invoked by admin clients for managing the Wookie server, e.g. for managing whitelist entries or widget access policies.

Authentication

By default the Admin REST API is secured using the Admin security restrictions defined in web.xml. This means that typically the client needs to have authenticated with the server using the admin user credentials.

Response formats

Clients may request a response in either XML or JSON by setting the appropriate request content type. (If it is not possible to specify a content type in the request, clients may use the optional "format" parameter to specify a content type override.)

Whitelist

Action	Request	Example	Description
	GET {wookie}/whitelist		Returns all whitelist entries, consisting of an identifier and a URL. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication
	POST {wookie}/whitelist/ {param:url}		Creates a new whitelist entry with the URL provided using the *url* parameter. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.
	DELETE {wookie}/whitelist/{id}		Deletes the whitelist entry specified by *id*. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.

Widget Access Request Policies (WARP)

Action	Request	Example	Description

	GET {wookie}/warp {param: widgetId}		GET {wookie}/warp {param: widgetId}
	GET {wookie}/warp/{id}		Returns the access policy specified by *id*. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication
	POST {wookie}/warp/ {param:widgetId, origin, subdomains}		Creates a new policy with the details provided. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.
	PUT {wookie}/warp/{id} {param: granted}		Updates the policy specified by *id* with the status of *granted* if the *granted* parameter is set to "true", otherwise sets the status of the policy to *not granted*. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.
	DELETE {wookie}/warp/{id}		Deletes the policy specified by *id*. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication.

Widgets

Action	Request	Example	Description
	POST {wookie}/widgets {file}		Posts a widget file to the server; this is identical in behaviour to dropping a ".wgt" file into the Wookie deploy folder. This method requires authentication using a widgetadmin role, e.g. using HTTP Basic authentication