Project no. FP6-028038

PALETTE

Pedagogically sustained Adaptive LEarning Through the exploitation of Tacit and Explicit knowledge

Instrument: Integrated Project

Thematic Priority: Technology-enhanced learning

D.IMP.07 – Final version of PALETTE registry and delivery framework

Due date of deliverable: September 30, 2008

Actual submission date: October 15, 2008

Start date of project: 1 February 2006 Duration: 36 months

Organisation name of lead contractor for this deliverable: CRP-HT, CTI

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
R	PUBLIC	**PU**

Keyword List: PALETTE Services, PALETTE Services Registry Framework, PALETTE Registry User Interface, PALETTE Services Description Schema and Documents, PALETTE Service Registration, PALETTE Services Delivery Framework, PALETTE Services Portal, PALETTE Services Browser, Widgets, Inter-Widget Communication, Widget Authentication

Responsible Partner: CRP-HT, CTI

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
0.11	05/09/2008	1st DRAFT	Packaged by YNA. Including contributions from CRP-HT, EPFL and UNIFR.
0.13	09/09/2008	2nd DRAFT	YNA: added contributions from CTI, and changes from UNIFR; reorganised the whole document; changed summary and introduction.
0.14	11/09/2008		JBO: somes modifications and adds in the portal section
0.15	15/09/2008	3rd DRAFT	All: Removed task 2 related parts, which will be moved to D.IMP.08; changed plan; change of introduction. Section 3.2 and conclusion changed. PSRF section updated.
0.16	16/09/2008	Final Draft	YNA: accepted modifications, added conclusion paragraphs from CRPHT and UNIFR; section 3.2.6 moved to conclusion; conclusion re-writting
1.0	16/09/2008	1st release, ready for review	CTI: added PSRF conclusion part
1.3	06/10/2008	Release Candidate	Integrated changes according to reviewers remarks
1.4	10/10/2008	Release Candidate	Integrated changes according to SC remarks

Deliverable manager

- Yannick Naudet, CRP-HT
- Nikos Karousos, CTI

List of Contributors

- Jérôme Bogaërts, Jean-David Labails, Yannick Naudet, Alain Vagner, Marie-Laure Watrinet, Géraldine Vidou (CRP-HT)
- Stéphane Sire (EPFL)
- Sami Miniaoui (UNIFR)
- Manolis Tzagarakis, George Gkotsis, Nikos Karousos, Nikos Karacapilidis (CTI)

List of Evaluators

- Stéphane Sire, EPFL
- Amaury Daele, UNIFR

Summary

This document presents the final version of the PALETTE Services Registry and Delivery Frameworks, which will be made available for communities. The PALETTE Services Delivery Framework (PSDF) aims at providing potential users with an easy overview, access and management facilities to the palette of services resulting from the PALETTE project. The PSDF is linked to the PALETTE Services Registry Framework (PSRF), which provides a registry where all PALETTE services are referenced. It provides user-oriented interfaces to the PSRF: the PALETTE Services Browser (PSB) and the PALETTE Services Portal (PSP). A specific section is dedicated to inter-widget communication in the PSP, which is a main achievement allowing composition of services at the presentation level. Last changes and enhancements on these frameworks, as far as their first

version is concerned (see D.IMP.04, D.IMP.05 and D.IMP.06), are also presented in this document. In particular a usability study is proposed and taken into account in the enhancements.

1 Introduction

This document presents the final versions of the PALETTE Services Registry Framework (PSRF) and of the PALETTE Services Delivery Framework (PSDF), which is provided to communities aiming at enhancing their efficiency thanks to the palette of services we offer them. While presenting the last features and enhancements of the frameworks components already described for most of them in the deliverable D.IMP.06, we take here an end-user oriented approach. All along the development life-cycle of the provided tools, we have adopted a user-centred design and a particular attention has been given to ergonomics and usability, to which specific sections are dedicated in the description of each tool.

The targets of the present report are the potential users of PALETTE's palette of services, both standard users and developers aiming at making the integration in the CoP's IT environment. It will be useful to developers of PALETTE services, to get to know the APIs that are available to advertise their services and to integrate their services through the provision of widgets. Finally, it may also be useful for CoP mediators, mainly to get a description of the latest version of the service browser and to have an idea of the new possibilities of the Portal.

The final version of the PSRF has been released after the consideration of a set of critical issues that concern the service description and registration:

- Definition of a common accepted PALETTE Service description schema, which constitutes the basis for all the PALETTE Service description documents.
- Definition and implementation of the set of the provided PSRF methods.
- Acceptance of the communication protocol.
- Fixing of bugs and improvement of performance.

In the context of PSRF usage, a web based User Interface for the registry has been also designed in order to provide a human-friendly way for the PALETTE partners to register and publish their PALETTE services. Both the PSRF description and information about the Registry UI are presented in this document.

The final PSDF provides the following functions, providing a great flexibility in the exploitation of PALETTE services with different levels of integration in the existing architecture of a CoP and answering to different needs:

- PALETTE Services Portal (PSP): A web portal allowing CoPs or CoPs' members to compose a personalised interface to PALETTE services, including aggregation and composition of those services;
- PALETTE Service Browser (PSB): A (web) graphical interface enabling CoPs' members to search and use new services in the PALETTE services registry.

The remainder of this document details the lasts versions of the PSRF (section 2) and of the PSDF components, the registry, the service browser, and the portal (section 3). Components are described focusing on enhancements and changes since the intermediary release presented in D.IMP.06. Section 3.2 details the mechanisms provided for inter-widget communication, allowing services composition at the presentation level. Section 3.5 is dedicated to experimentations in CoPs and usability tests, with the recommendations provided as a result. Finally, section 2 concludes and gives hints for evolution perspectives.

1.1 Acronyms

- API: Application Programming Interface
- CoP: Community of Practice
- CSUQ: Computer Science Usability Questionnaire

- CSS: Cascaded Style Sheets
- DBMS: DataBase Management System
- DOM: Document Object Model
- GUI: Graphical User Interface
- HTML: HyperText Markup Language
- HTTP: HyperText Transfer Protocol
- HTTPS: Hypertext Transfer Protocol over Secure Socket Layer
- INFOQUAL: Information Quality
- INTERQUAL: Interface Quality
- ISO : International Standards Organization
- IT: Information Technology
- MVC: Model View Controller
- PSB: PALETTE Service Browser
- PDM: Participatory Design Methodology
- PSP: PALETTE Services Portal
- PSDF: PALETTE Services Delivery Framework
- PSRF: PALETTE Services Registry Framework
- REST: Representational State Transfer
- RSS : Really Simple Syndication
- SOAP: Simple Object Access Protocol
- SYSUSE: System Usefulness
- SUS : System Usability Scale
- UI: User Interface
- URI : Universal Resource Identifier
- URL : Uniform Resource Locator
- UWA : Universal Widget API
- WADL : Web Application Description Language
- WAM: Widget Authentication Mechanism
- WRWD : The W3C Widgets 1.0 Requirements working draft
- WSDL : Web Service Definition Language
- XHTML: Extensible HyperText Markup Language
- XML: Extensible Markup Language

1.2 Reading conventions

References are listed at the end of the document in section 5. References by author, like (Fielding, 2000), appear in a bibliography in section 5.1, while references by code, like (URL: ALU), appear in a "webography" (list of Web links), section 5.2.

2 The PALETTE Services Registry Framework

PSRF has been designed in the context of the PALETTE Service Platform Architecture (as it has been presented at D.IMP.04). It is a service description repository, which is responsible to maintain and publish description records of all the available PALETTE services. In particular, the main PSRF tasks are:

- to publish the available services by registering service description documents to the registry;
- to modify the above service descriptions;
- to search and discover services, and
- to handle client authentication.

The first implementation of PSRF was released at the second year of the PALETTE project. However, some changes concerning the provided API have been required during the interoperation with the PALETTE Service Delivery. Furthermore, the service description schema was modified according to both service providers' suggestions and some performance issues. Thus, a new version of PSRF was finally produced.

This section reports on the final version of the PSRF. A complete list of the modifications, additions and improvements that have been applied in PSRF since its first version is also presented. Finally, the section provides some general information about the PSRF user interface.

2.1 The Final Version of PSRF

PSRF consists of the "Access and Identity Mechanism" and the "PALETTE Service Registry". Both modules are being implemented in order to securely store, publish and manage the PALETTE Service Description Records that are produced by the PALETTE Service Providers. Despite the fact that each of the modules has a specific role, both modules are strongly cooperating in internal layer. In PSRF, an integration library combines the two different modules and provides to the end-users (service providers, applications and PALETTE members) a superset of methods through the PSRF API.

The **Access and Identity** mechanism module is responsible for managing the identity repository and for executing the required mechanisms that will authenticate users as well as applications. Through this mechanism, the service providers will be able to register themselves in the PALETTE registry and the registry administrator will have the ability to manage the registered accounts. The main entity of this module is the identity entity called "Account". An "Account" can be related to humans or applications. Currently, PSRF supports three different types of accounts, namely: Administrator, Service Provider and Simple User or Application. When someone is being authenticated as an Administrator, all the PSRF API methods become available. Service Providers can create and modify registry records into the framework, while simple users (or applications) have view and search privileges only.

The PSRF API methods that concern the Access and Identity mechanism are the following (more technical details can be found at (URL:PSRF):

Method	Description
IdentifyAccount [name, password]	Identifies an Account by name and password. If the identification succeeded, returns 1 else 0. If operation fails, operation error is returned.
LoadAccountByID [id_account]	Loads an Account from Registry by its id. This operation requires the Administrator Credentials for the requestor. Returns the Account filled up with the Account data. If no data then it returns 0. If operation fails, operation error is returned.

LoadAccountByNamePass [name, password]	Loads an Account from Registry by its Name and Password. Returns the Account filled up with the Account data. If no data then it returns 0. If operation fails, operation error is returned.
CreateAccount [name, password, email, affiliation, type]	Creates an Account in Registry. This operation requires the Administrator Credentials for the requestor. Returns the id of the created Account as the ResultCode Number. If operation fails, it returns a negative operation error.
UpdateAccount [id_account, name, password, email, affiliation, type]	Updates an Account in Registry. This operation requires the Administrator Credentials for the requestor. Returns 1 as the ResultCode Number. If operation fails, it returns a negative operation error.
DeleteAccount [id_ account]	Deletes an Account from Registry by its id. This operation requires the Administrator Credentials for the requestor. The account will be deleted only if there are no integrity constrains. Returns 1 if the Account was deleted. If operation fails, operation error is returned.
CheckEmail [email]	Check for the existence of an email. This operation requires the Administrator Credentials for the requestor. This method is useful in case an application wants to pre-check for the existence of the email during the registration procedure. Returns 1 if emails exists, otherwise 0. If operation fails, it returns a negative operation error.
CheckName [name]	Check for the existence of an Account Name. This operation requires the Administrator Credentials for the requestor. This method is useful in case an application wants to pre-check for the existence of the Account Name during the registration procedure. Returns 1 if Name exists, otherwise 0. If operation fails, it returns a negative operation error.
CheckEmail [email]	Check for the existence of an email. This operation requires the Administrator Credentials for the requestor. Returns 1 if emails exists, otherwise 0. If operation fails, it returns a negative operation error.
CheckName [name]	Check for the existence of an Account Name. This operation requires the Administrator Credentials for the requestor. Returns 1 if Name exists, otherwise 0. If operation fails, it returns a negative operation error.
LoadAccountList [name, password]	Loads a List of all active Accounts of the registry. This operation requires the Administrator Credentials for the requestor. Returns 1 if all ok. If operation fails, it returns a negative operation error.

The PALETTE Service Registry concerns the mechanism that holds the descriptions of all published PALETTE services. In particular, the requestors are able to search, locate and finally use the PALETTE services descriptions during design time in order to generate the necessary "stubs" for communicating with other PALETTE services and integrating them with their own. These descriptions (registry records) are stored internally in an XML format that is complied with a specific PALETTE Service XML Schema. This particular schema can be found at (http://150.140.18.39:91/psrf/img/ServiceDescriptionSchema.xsd). In our approach, a PALETTE

service can be a web service, a web application, or a non-web application. The PALETTE Registry provides to the PSRF API a rich set of operations that can store, manipulate, publish and search PALETTE Service Description Records (more technical details can be found at (URL:PSRF). These operations are described below.

Method	Description
CreateService [account_id, status, xml_content]	Creates a Service Description in Registry. This operation requires the Administrator or the Service Provider Credentials for the requestor. Returns the id of the created Service as the ResultCode Number. If operation fails, it returns a negative operation error.
UpdateService [id_service, status, xml_content]	Updates a Service Description in Registry. This operation requires the Service Provider Credentials for the requestor. Service providers can update only service descriptions they have created. Returns 0 ResultCode Number. If operation fails, it returns a negative operation error.
DeleteService [id_service]	Deletes a Service Description from Registry. This operation requires the Service Provider Credentials for the requestor. Service providers can delete only service descriptions they have created. Returns 0 ResultCode Number. If operation fails, it returns a negative operation error.
LoadServiceByID [id_service]	Loads a Service Description Record from the Registry. Returns a Service Description together with 0 as ResultCode Number. If operation fails, it returns a negative operation error.
PublishService [id_service]	Publishes the Service Description. (Makes it available to the public) Returns 0 as ResultCode Number. If operation fails, it returns a negative operation error.
UnpublishService [id_service]	Unpublishes the Service Description. Returns 0 as ResultCode Number. If operation fails, it returns a negative operation error.
LoadServiceList	Loads a List of all active Services of the registry. Returns 1 if all ok. If operation fails, it returns a negative operation error.
LoadServiceListByAccount [id_account]	Loads a List of all active Services of the registry that have been created by an account. This operation returns both publish and not publish services. Returns 1 if all ok. If operation fails, it returns a negative operation error.
SearchServices [name, provider, category, type, contributor, keyword, description, orderByPublicationDate]	Loads a List of all active Accounts of the registry. This operation requires the Administrator Credentials for the requestor. Param elements are mandatory. Params may not be filled. When more than one param have values they are linked with the operand "AND". The string matching is based on a substring case insensitive text search. Returns 1 if all ok. If operation fails, it returns a negative operation error.
SearchServicesGlobal [key]	Searches all published Services of the registry using a global search given a keyword. Returns 1 if all ok. If operation fails, it returns a negative operation error.

PSRF is operating as a web service by itself. It is a stand-alone server that sends and receives messages over the HTTP protocol. It can be accessed through REST methodology (it is listening to HTTP POST) and can be easily converted to a classic SOAP-based web service. In particular, PSRF supports two kinds of request messaging. The first one (XML messaging) uses XML formatted requests and responses, while the second one is based on interchanging simple name-value pairs into the message body. A real example of the two different requests is displayed bellow. Although the communication protocol is the HTTP, the body of the request message may be structure in two different ways.

XML Request	**Name-Value Request**
<?xml version="1.0" encoding="utf-8"?> <Request> <Authentication> <Name>nikos</Name> <Password>nikos</Password> </Authentication> <Operation> <Name>LoadServiceByID</Name> <Params> <id_service>12</id_service> </Params> </Operation> </Request>	Name=nikos&Password=nikos&S_IDService=12& OpName=LoadServiceByID

The development of PSRF was based on Microsoft's .NET v2.0 framework using the MS Visual Studio Development Platform and the C# programming language. The storage layer is supported by Microsoft SQL Server 2005 DBMS which provides native functionality for XML document storage and retrieving. The hosting and publishing of the PSRF API takes place through the Microsoft Internet Information Server. Finally, an available client of the PSRF API (http://150.140.18.39:91/psrf) has been developed using the PHP programming Language.

2.2 Using PSRF.

The provided API of PSRF can be used in order to integrate the registry's functionality with any external service browsing application. In the context of PALETTE, the Service Delivery Framework (see chapter 3), through which one can search and find information about all available PALETTE Services and widgets, is based on PSRF. Thus, through the invocation of PSRF API methods, the Service Delivery can provide all the aforementioned functionality. In particular, when someone wants to find detailed information of a particular service, he/she can access the entire service description document in which both explanatory details and URL links to WADL or WSDL descriptions can be easily found.

PALETTE partners can use PSRF for registering their tools and services through their accounts. Currently, they can use the PSRF client pages for this purpose. However, the service management functionality will be provided through the Registry UI (see section 2.5) that is still under development.

Until now CTI, EPFL and CRP-HT have already created Provider Accounts in order to exploit the PSRF functionality. Moreover, a list of PALETTE services is already registered in the PSRF. These are:

1.CoPe_it!

2.Amaya

3.LimSee3

4.eLogbook

5.DocReuse

6.SweetWiki

7.BayFac

8.CoPe_it! search web service

9.SemanticFaq Portal

10.Semantic Retrieval Service

11.Ontology Management Service

12.Annotation Management Service

13.Semantic Engine Administration Service

14.PALETTE Service Registry

2.3 Current status and location information

PSRF is operating as a Web Service using the HTTP protocol at http://paletteregistry.cti.gr/service. At (URL:PSRF-SERV) (figure 1), some overall information about PSRF is given. Furthermore, a detailed PSRF API documentation together with some samples of service invocation are available at http://150.140.18.39:91/psrf/ (figure 2). The final (updated) version of PSRF is version **1.2,** which has been released at 20/08/2008.

Figure 1: The Palette Registry Web Site

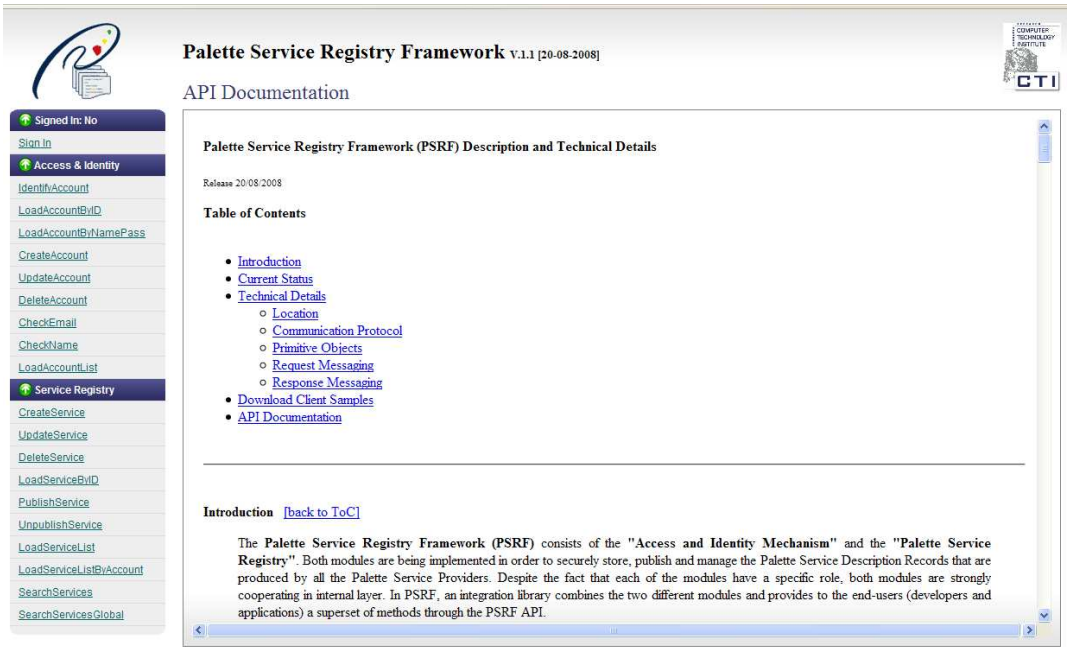

Figure 2: The Palette Registry API documentation Web Site

2.4 Modifications and Additions to the first release.

The required alterations and additions on the first version of the PALETTE Registry, which have been done according to the PALETTE users' needs, are presented below:

1.Addition of an extra *messaging mechanism* that uses *name-value pairs*: PSRF was initially operating by exchanging well-formed XML messages in the body of both http requests and responses. This has the advantage of handling the entire messages with a very easy way. Furthermore, it allows the transformation of the whole PSRF to a SOAP-based service without much effort. However, after the first real usage of the PSRF by the Service Browser UI, it was asked for the PSRF to also support a more RESTful approach of the messaging mechanism by using HTTP POST request pairs of names and values. Now, PSRF provides efficiently the whole set of its available methods in both ways.

2.Modification of the *PALETTE Service Description Schema*: The PALETTE Service Description Schema was firstly defined according to the D.IMP.04 guidelines for development after agreement between all the involved partners. However, some elements of the schema were added and some others removed after the implementation procedure, due to some reasons that concerned simplicity, usability and importance. Although these alterations took place after the first release of PSRF, all the existing service registrations were also updated according to the new schema. The final version of the schema is currently supported by PSRF and is available at http://150.140.18.39:91/psrf/img/ServiceDescriptionSchema.xsd.

3.Improvement of *searching capabilities*: Searching capabilities have been improved with extra functionalities. First, PSRF now supports search by keywords that can be matched in every point within the Service Description XML Document. Furthermore, case insensitive search is also supported by PSRF within the XML Document elements.

4.Improvement of *performance* for the existing methods: In order to improve performance issues during the requests and responses of PSRF, some changes took place in the design of both the storage and the interface layer. Thus, PSRF can now execute load and search operation in a very fast way.

5.Fixing of several *bugs* that occurred during usage.

2.5 The Registry UI

The Registry UI allows PALETTE partners to upload and manage their services' descriptions. It is based on the provided API of the PSRF and it assures that any partner will be able to securely handle all the registered PALETTE services.

The Registry UI operates in two different modes, namely the Administrator mode and the Service Provider mode. Within the former, the PSRF administrator can manage all PSRF Accounts and can accept or reject requests for creation of new Provider Accounts. Furthermore, the PSRF administrator can assist providers in the management of the registered service descriptions, as he/she has full permissions for the management of all PALETTE Services in PSRF. In the latter mode, Service Providers are able to create, alter, delete, publish and unpublish their services' descriptions through the "MyServices" control panel screen. In order to access the Registry UI, both administrator and service providers are required to authenticate themselves in the Registry UI login form, which is responsible to redirect users to the appropriate operational mode.

The development of the PSRF is based on Microsoft's .NET v2.0 framework using the MS Visual Studio Development Platform, the C# programming language and the ASPX dynamic web pages. The hosting and publishing of the PSRF API is taking place by the Microsoft Internet Information Server.

2.5.1 Current status and location information of the Registry UI

The Registry UI is currently under development. A first prototype of the UI will be temporarily released at http://150.140.18.39:85/ in order to be provided for a first evaluation by all PALETTE Partners. Then, after the feedback and customisation phase, the registry UI will be published at (URL:PSRF-SERV). Both the Registry UI and the Registry browser will be interconnected via direct links in order for PALETTE users to navigate easily between the two different interfaces of PSRF.

2.5.2 Future tasks of the Registry UI

Work on the Registry UI is currently focused on two critical tasks: the generation of a dynamic PALETTE service registration web-form, and the exploitation of the Service Registry UI.

In its first release, the Registry UI will support the upload of the entire PALETTE service description XML document in the form of an XML file that is firstly created and formatted by the users outside the context of the UI. In the registration procedure, the uploaded file will be validated against the PALETTE service description schema and will be inserted (or not) to the PSRF. However, in order to avoid both the external creation of the XML document and the validation procedure, the final version of the Registry UI will focus on the support of a dynamic web-form, in which all the necessary PALETTE service information will be filled as simple fields and will be checked on run-time.

The Service Registry UI will be exploited by all PALETTE partners in order to in order to both publish and manage their PALETTE services provided. CTI will have the role of the PSRF administrator in order to assist and coordinate the exploitation (according to the description of WP5 in IP3).

3 PALETTE Services Delivery Framework

3.1 Evolution

A complete refactoring of the PALETTE Services Portal (PSP) was undertaken in order to greatly improve its source code quality. Although it does not supply main functionalities for end-users, this step was necessary in order to allow much faster development of new major features.

The software architecture was completely recreated to totally fit to the Model-View-Controler (MVC) design pattern. The application is now built on a PHP MVC framework designed and developed at the CRP-HT. This new design aims at guiding the development process in a more object-oriented direction and settles a firm foundation for the future of the project. The source code is now entirely documented. All information relevant to the source code is at this time available as an XHTML-based referential generated by PHPDocumentor (URL:PHPD), for a more convenient and efficient reading.

The installation process is now easier than before because the software does not rely anymore on particular PHP/PEAR (URL:PEAR) libraries that are not included by default with PHP5. At present, the portal is quite built upon core PHP5 features.

Efforts were made on the internationalisation of the graphical user interface of the portal that is now provided by default in English, French and Dutch. In the future, every language the portal needs to be translated in, including languages with particular characters such as Chinese and Arabic may be supported. In summary, a full Unicode translation mechanism is provided and the way to add a new language is kind of trivial.

The portal also supports the newly designed inter-widgets communication mechanism. Currently, widgets are able to notify events to other ones, using “Unicast”, “Multicast” or “Broadcast” event firing mechanisms that are accessible for widget developers through the existing widget API.

Concerning the PALETTE Services Browser (PSB), a few enhancements have been achieved, centred on usability by providing new search functions by categories or tags and a specific widget that can be inserted in the PSP.

The next sections details all those evolutions, starting with the new features of the PSP, namely the inter-widget communication and the widget authentication mechanisms, and ending with the PSB.

3.2 Inter-widgets communication in the PSP

The portal realizes the visual integration of several services through the provision of a widget for each service. However, this is not enough to allow interaction between services to be triggered from the portal. For that purpose we have developed an inter-widgets communication mechanism that is illustrated in the following sections after a short presentation of some scenarios.

3.2.1 Scenarios involving inter-widget communication

When a news widget receives a news item about a particular company, it could tell a stock quotes widget to display any changes in that company's share price. This example is taken from the requirement 33 of the W3C Widgets 1.0 Requirements working draft (URL:WRWD). It illustrates a first type of inter-widget communication that we can call widget state coupling.

Widget state coupling is required whenever a widget holds some data or some state information, which can be updated, either from the system, or as a consequence of a user action. For instance, the action of the user could be to push a button displayed next to a result list displayed in a widget with the effect of selecting a particular result item, which could then be automatically transmitted to another widget being able to do some treatments with that item.

Another scenario that we call the drag and drop scenario is an extension of the drag and drop operation, which is natural in direct manipulation user interfaces such as operating system desktops.

Figure 3 below is a mockup illustrating how a drag and drop operation could be use to drag a particular result from a Common Repository Browser widget and to drop it onto an XTiger Template

Preview Widget to invoke a pre-visualization service showing the XTiger template file with different preview styles which can be selected before from a popup menu. This example is quite realistic as both widgets are currently under development by PALETTE partners.

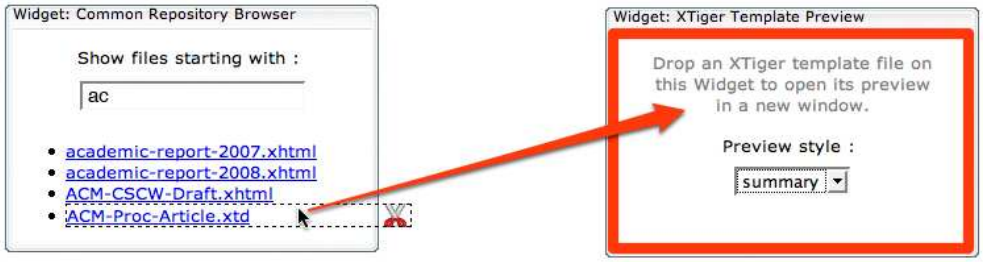

Figure 3: example of Drag and Drop operation between two widgets.

Another example is a contact widget displaying the list of contacts of the user with eventually some online status information. Such a list is a shortcut to select a user, which can then be used to perform all kinds of operation into other services. For instance, if a widget is available for negotiating meeting arrangements, it may be convenient to drag and drop the users to invite to a meeting directly from the contact widget to the meeting calendar widget.

3.2.2 Design issues for an inter-widget mechanism

We had to solve four major issues while designing an inter-widget communication mechanism.

The first issue concerns the identification of the events, which are transmitted to the widgets. These events can be named (or typed) with different schemes. It seems imperative that a good naming scheme should allow decoupled widgets to address events from each other, without knowing their implementation details. This is required for instance to develop mechanisms such as publish-subscribe between widgets.

The second issue concerns the transmission of data associated with an event occurrence. Basically two mechanisms can be retained:

- **by value**: the data is directly encoded within the message of an event occurrence, which is transmitted from a source widget to a target widget.
- **by reference**: only a means to retrieve the data is transmitted, for instance in the form of a URI Reference which can be de-referenced to obtain the data.

The third issue concerns the style and the syntax of the "wiring" between source widgets and target widgets. Several styles can be chosen to allow direct wiring between a source widget and a target widget, or to allow wiring between a source widget and a set of target widgets, limiting or not the wiring to subset of event types and/or eventually conditions on event data. For each style different syntaxes can be chosen, either purely procedural consisting of JavaScript functions in our context, purely declarative, or a mix between both. It is to notice that in the case of the drag and drop scenario, it is also necessary to provide a syntax to identify the widget user interface elements which can be dragged and to bind them with event types and event data.

Finally, the last issue concerns the user interface integration of the inter-widget communication mechanism.

The state-coupling scenario requires that users must be able to anticipate which widgets influence each other when they are instantiated together on the same portal. Here the designer faces with two solutions: either to let users manage explicitly the coupling between widget states, or to handle it automatically. The second case seems the easiest from a user's point of view. However, for some cases when the users wouldn't like to allow a coupling between two widgets states, it seems necessary to provide a portal with a mechanism to "decouple" two widgets, and to "undo" that decoupling.

The Drag and drop scenario requires that users be able to detect when a drag operation is permitted from a source widget and to identify the target widgets. The first feedback is usually provided through a cursor change when flying over a drag source, and with an highlighting when flying over a potential target. This allows the provision of a generic mechanism that would work at the portal level.

3.2.3 Messaging API for inter-widget communication

As a starting point, we have designed first an event notification model allowing widgets to communicate with other ones through message-based widget event. This mechanism allows widgets running in separate and isolated contexts to communicate with each other using event messages that are transmitted by the widget engine from a source widget to one or multiple target widgets.

A message transmits the name of an event and its associated data *by value*. The event names are identified with conventional URIs. This will allow to formalize the semantic structure of declared event types either by extending the widget manifest file to contain a semantic description of the event data, or by allowing to de-reference the event type URI to retrieve its semantic description.

Widget developers can choose between three kinds of event firing: "Unicast", "Multicast" and "Broadcast". For the first type, the message and the data associated with the event are sent to a single target widget. On the other hand, a "Multicast" event firing sends event messages to a list of identified widgets. Finally a "Broadcast" event firing will notify all widgets listening for the right event type that an event has been raised. Figure 4 illustrates the mechanisms.

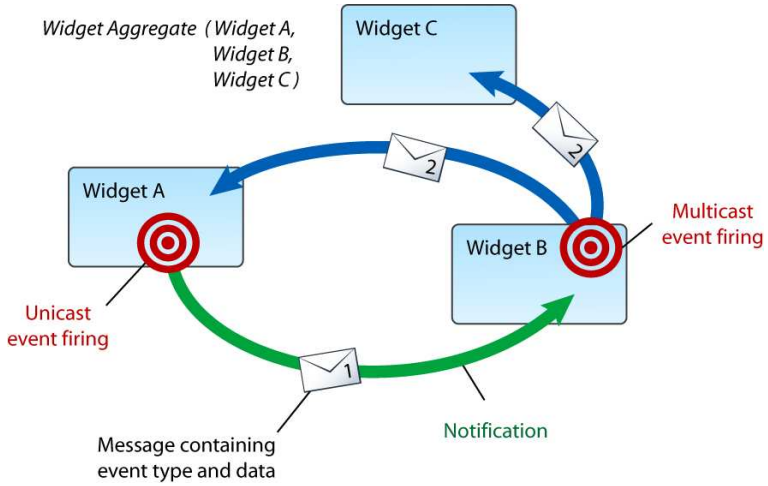

Figure 4: Widget events firing.

The messaging API allows the developer of a widget to notify events to other widgets using a Javascript procedural API based on the methods detailed in the table below. These methods were added to the existing Widget API. They are accessible through the window.widget JavaScript object that is injected at runtime into the runtime context of every widget instance. These new methods were designed in order to provide a low-level messaged-based event API that might be used to build other types of widget inter-communication such has Drag and Drop of objects between widgets.

Method	Description
widget.fireWidgetEvent(string: ***eventType***, Object: ***data***, string: ***target***)	Fire an event of a given ***eventType*** in order to transmit some ***data*** in the transmitted message, for a particular ***target*** widget alphanumeric identifier. If the ***target*** is set to *null*, the event will be notified to every widget instance listening for this event type.
widget.addWidgetEventListener(string: ***eventType***, Function: ***eventHandler***,	Listen to ***eventType*** events that are coming from a specified alphanumeric widget identifier list defining the

Array: ***acceptedSources***)	***acceptedSources***. When events are fired, the function defined by ***eventHandler*** is executed. If the ***acceptedSources*** list is set to null, the widget will accept be notified of this type of event from any source widget.
widget.removeWidgetEventListener(string: ***eventType***, string: ***source***)	Stop to listen to a given *eventType* for a particular ***source*** widget identified by an alphanumeric widget identifier. If the ***source*** is set to *null*, the widget will completely stop to listen to this kind of events.

Widget developers are also able to choose which kind of events their widgets will listen to. This feature was designed in order to reduce dramatically the message traffic on the portal. If an event of a given type is fired but no widgets are currently listening to it, the related message will actually never be sent through the portal and will simply be ignored. Developers can also take the decision to listen for a particular event type fired by a particular widget, or simply accept all events of a given type from any widget.

Moreover, the portal is also able to fire events to widgets listening to a particular portal related event. For instance, a widget can listen for an event that aims to warn that a widget changed its state from a user point of view. The portal could fire this type of event when a widget is maximized, minimized or simply removed from the user interface. The different types of events that are generated by the portal are described in the table below:

URI	Description	Data transmitted
http://PALETTE.ercim.org/ns/event/maximized	The "Widget maximized" event is sent when a widget is toggled to a maximized state.	▪**widgetId**: the alphanumeric identifier of the widget that was maximized.
http://PALETTE.ercim.org/ns/event/minimized	The "Widget minimized" event is sent when a widget is toggled to a minimized state.	▪**widgetId**: the alphanumeric identifier of the widget that was minimized.
http://PALETTE.ercim.org/ns/event/removed	The "Widget remove" event is sent when a widget is removed from the user interface.	▪**widgetId**: the alphanumeric identifier of the widget that was removed.
http://PALETTE.ercim.org/ns/event/added	The "Widget added" event is sent when a widget is added on the user interface.	▪**widgetId**: the alphanumeric identifier of the widget that was removed.

3.2.4 Specification of a Drag and Drop API for inter-widget communication

In a second time, we have specified a simplified API specifically targeted at the drag and drop scenarios. We have currently made the following design choices, very similar to the choices made for the messaging API above :

- drag and drop events are named with conventional names based on URIs ;
- drag and drop data is transmitted by value, although we encourage developers to transmit only a URI string, which could be used to get more data by de-referencing this URI ;
- the API is procedural, with Javascript methods for defining the "wiring" between widgets, and for binding an event data type URI and its data with a DOM element of a widget.

The reason to encourage developers to use URI strings as event data is that in most cases, drag and drop items represent resources such as files hosted within PALETTE services.

The third choice allows reusing the addWidgetEventListener method of the existing messaging API to subscribe to drag and drop events. This method of the widget object has been completed with a bindWidgetToDropType method to allow a widget to declare its interests in some drag and drop event types. This allows the drag and drop part of the widget engine to give appropriate feedback when the user is dragging data on top of a widget frame.

As an example, the code extract below declares that a widget can receive drag events of type 'http://PALETTE.ercim.org/ns/dnd/file/xtiger-template'. It also indicates that a Javascript inline method must be called when such an event occurs. The method will be passed an event parameter by the widget drag and drop engine, through the messaging API as described above.

```
widget.bindWidgetToDropType ('http://PALETTE.ercim.org/ns/dnd/file/xtiger-template');
widget.addWidgetEventListener('http://PALETTE.ercim.org/ns/dnd/file/xtiger-template',
        function(event){
                …
        },
        null);
```

We have also extended the existing messaging API with a method described below to dynamically declare drag event sources in a widget. That method allows associating a drag event type together with some data with a DOM element of the widget. It is also possible to specify a tooltip text that will be displayed during a drag action. As a consequence of a call to that method, the widget engine modifies the DOM element so that it displays a proper feedback on mouse over and it triggers a drag action when the user drags it.

Methods for Drag and Drop	**Description**
widget.bindWidgetToDropType (string: ***eventType)***	Declares that a widget is a drop target for drag and drop events of a given ***eventType***. This will inform the drag and drop widget engine to display a proper feedback when dragging some compatible data other the widget frame.
widget.addDragData (Element: ***domElement***, string: ***eventType***, Object: ***data***, string: ***tooltip)***	Transforms a ***domElement*** of the widget tree into a draggable element that triggers a drag event of a given ***eventType*** with an associated ***data*** and that displays a ***tooltip*** message during drag. The widgets that registered for the drag event of type ***eventType*** with ***bindWidgetToDropType*** will be notified if the user drops the widget element onto them.

The drag and drop API could be extended in the future, with the introduction of a declarative API. That would allow using the widget manifest file to declare the potential "wiring" of a widget with the other widgets instantiated in the same portal. This could be migrated to a widget preference panel for being directly edited widget instance by widget instance. This supports the design of graphical user interfaces to allow users to set up widget wiring by hands if they want to overwrite the defaults.

3.2.5 Related work

The two APIs described in this section have been inspired by a careful look at the state-of-the-art.

The Java portlet specification version 2 also called JSR 286 (Hepper, 2006) encourages developers to name events in a hierarchical manner using the dot '.' as separator. This allows portlets to use a "*" character to declare that they are willing to process any event whose name starts with the characters before the "*" (such as "com.acme.*"). Event data are passed *by value*, as Java objects. JSR 286 uses a declarative API, for instance to declare the data type for a given event name into the portlet deployment descriptor file, or to declare the event types supported or published by a portlet. It also uses a procedural API to handle the received events in a "processEvent" handler method or to transmit new events. It is to notice that event reception and event emission can only occur at some special points in the execution flow (i.e. when processing client requests).

The HTML 5.0 draft recommendation describes a cross-document messaging API (URL:HTML5). This could work for communication between widgets too, as in most portals, including the PALETTE portal, each widget has its own document. The HTML 5.0 doesn't distinguish between an event type and an event data. An event is defined by a string message. The API defines a Javascript method, "postMessage", to create an event occurrence and to post it to a target document. Target documents are notified if they have registered an event handler with the "addEventListener" method. This API only allows a "unicast" type of communications where the sender must know the target in advance. To our knowledge the HTML5 cross-document messaging API hasn't been implemented yet in any browsers.

The Universal Widget API (URL:UWA) is the Netvibes for building portable widgets. These widgets can run on multiple portal environments such as Netvibes, iGoogle or Live.com. They can also be run on desktop widget runtime engines such as Apple Dashboard or Opera widgets through a special widget provided by Netvibes. The UWA does not have yet any specific methods and syntax to allow inter-widget communication. However, a few generic events are generated by the widget engine during the widget life cycle and they can be notified to the widget which just needs to implement some corresponding event handler methods. The event handlers are functions with predefined names such as "onLoad", "onResize "or "onUpdateTitle".

3.3 Widgets authentication mechanism on a widget's side

Because it could be painful for a user to provide over and over again its identifiers for each PALETTE service he uses, and because these identifiers must be safely transmitted through the network, the "Widget Authentication Mechanism" enables the PALETTE Services Portal (PSP) to automatically authenticate widgets and their users against PALETTE services. It is able to establish a trust relation between a given widget installed on the PSP and a PALETTE service, but also to automatically transmit the identity of a user using this widget to the relevant service in a secure way.

3.3.1 Overview

The Widget Authentication Mechanism (WAM) was designed following two pre-established main goals:

- Provide a feature allowing establishing relations of trust between widgets hosted on the PALETTE Portal and PALETTE services.
- To make the portal a single entry point for the entire system. If users are authenticated on the PALETTE Portal, they must be authorized to consume PALETTE services.

To establish relations of trust between widgets and PALETTE services but also authenticate users invoking services, we decided that they had to share a secret that we can rely upon. To do so, widgets and services will share a secret "service key".

The shared Service Key will enable the portal to encrypt data identifying a user using a particular widget (the unique identifier of the user on the portal). When the service is invoked, the portal inserts on the fly the identifier of the user, and the encrypted version of it in the service request.

When the service receives this request, it will decrypt the encrypted data using the same key that was used for encryption. If the decrypted data matches the user identifier, the service is certain that the message comes from the portal on behalf of an authenticated user.

The following example (Figure 5) represents an authentication process for the Bayfac widget:

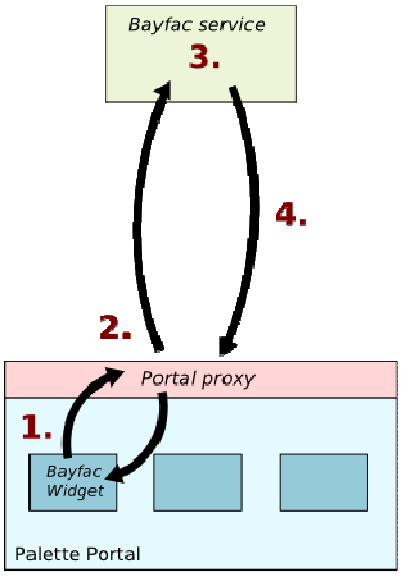

Figure 5: Widget Authentication Process.

1.The widget performs a request to the Bayfac Web Service.

2.The request is intercepted by the proxy of the portal. The identifier of the user and the encrypted version of this identifier are inserted on the fly into the service request.

3.The Bayfac Web Service decrypts the encrypted user identifier with its instance of the shared service key. If the decrypted identifier matches the clear one, it accepts to handle the request on behalf of the user represented by the identifier.

4.The Web Service sends its response to the proxy of the portal, which will transparently transmit the response data to the Bayfac widget.

3.3.2 Shared secret keys

The secret keys shared between widgets and PALETTE services are 256 bit keys involved in a cryptographic symmetric key algorithm. The algorithm used by the portal and services to encrypt/decrypt data using shared secret keys is the Rijndael algorithm (URL:APR).

Because of security issues, the keys will not be stored in the widget implementations but generated and managed by the Portal at widget installation time. Indeed, having an access to the source code of the Widget implementation should be enough to get the key and break the secret.

3.3.3 Activate the Widget Authentication Mechanism

For each Widget installed on the portal, which wants to use the Widget Authentication Mechanism, the portal will generate and manage a key that has to be known by the PALETTE service relevant to the Widget implementation.

To declare that a widget will use the WAM, a simple declaration in the manifest of the widget is enough, using the new `widget_authentication` element of the PALETTE Widget Specification. This can be done by adding the following line within the widget element of a widget manifest:

```
<widget_authentication>enabled</widget_authentication>
```

By inserting `a widget_authentication` element containing the enabled value in a widget manifest, the portal will automatically generate a secret key for the relevant widget during the installation process.

When the installation is successfully done through the administration section of the portal, a yellow key-shaped icon will be displayed near the widget meaning that the WAM was successfully activated. By clicking this icon, the portal administrator will be able to visualize the secret key (encoded in base64) that must be shared by the portal and a given PALETTE Service.

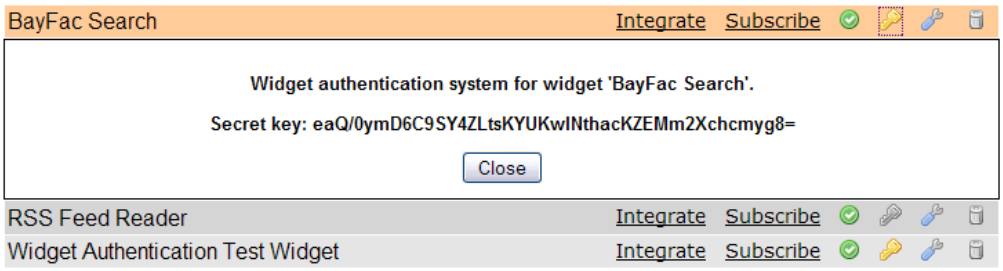

Figure 6: The widget authentication secret key for the Bayfac widget in the portal user interface.

3.3.4 Widget Authentication Mechanism on the PALETTE service's side

When the PSP generates a secret key for a widget being installed, this key must be transmitted manually to the targeted PALETTE Service. Although a mechanism allowing the PSP to transmit a generated key to the relevant service may be an improvement of the system, such feature would force PALETTE Services to implement a web service for handling the reception of the secret key.

In order to support the WAM, PALETTE Services must implement the decryption of the encrypted user identifier by using the shared secret key. When a PALETTE service is requested through a widget, the proxy of the portal injects on the fly the identity of the user in the HTTP request. In this process, the HTTP basic authentication (RFC2617) feature is used to transmit the identity data. The value of the *username* contains the identifier of the user, and *password* is the encrypted version of the unique identifier of the user.

When a PALETTE service supporting the WAM receives an HTTP request, it has to:

1. Retrieve the *username* and *password* values contained in the Authorization HTTP header of the request.
2. Decrypt the *password* value by using the previously shared secret key.
3. Check if the decrypted *password* value matches the username value. If the matching is effective, the PALETTE Service assumes that the request is coming from a trusted entity (the PSP), on behalf of a particular user identified by *username*.

Regarding the Rijndael decryption algorithm, we created a simple PHP library allowing developers to encrypt/decrypt data efficiently using a particular secret key. Here is a simplified sample of code used on the Bayfac Search Service that now supports the Widget Authentication Mechanism:

```
// Portal - Agent's secret.
$secret = base64_decode('FgdSYJBBbUD/Sxo+YJDgeECwB31nhxMEufKHHdVrvKw=');

// The Rijndael.class library contains methods to encrypt/decrypt data
// using the Rijndael algorithm.
$crypto = new Rijndael();
$message = $_SERVER['PHP_AUTH_USER'];
$cryptedMessage = $_SERVER['PHP_AUTH_PW'];
$decryptedMessage = $crypto->decrypt($cryptedMessage, $secret);

if ($decryptedMessage == $message)
{
      // Handle the request.
        // …
```

```
}
else
{
      // Reject the request.
      header('WWW-Authenticate: Basic realm="Bayfac"');
      header('HTTP/1.1 401 Unauthorized');
      exit;
}
```

3.3.5 Identity transmission

According to "The laws of Identity" (URL:TLOI), *Technical identity systems must only reveal information identifying a user with user's consent*. Because the portal reveals the identity of a user to a trusted third-party service when authenticated widgets are used, users are asked to consent this revelation each time that such a widget is added to their graphical user interface.

Figure 7: Identity transmission's confirmation box.

Moreover, users of the portal must be able to know at any time when widgets transmit their unique identifier through trusted service requests. To allow them to be aware that a widget is currently using the Widget Authentication Mechanism, authenticated widgets will bear a key icon as on the following figure.

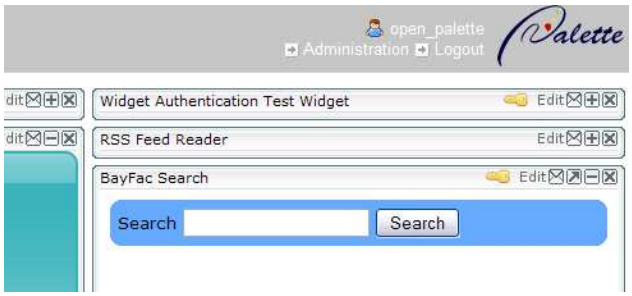

Figure 8: Widget Authentication icons in title bars.

3.3.6 Security

Although the risk is minimal, HTTP requests and responses exchanged by the portal and PALETTE services that contain data relevant to users (their identity) could be intercepted using a "Man in the middle" attack. To avoid such kinds of attacks, we recommend that services are accessible and invoked through HTTPS.

3.3.7 Technical grounding and implementation choices

In the design phase of this mechanism, we tried to build an authentication protocol API that was secure and with a good user experience.

At first we evaluated HTTP Basic authentication (URL:RFC2617), which is a classical authentication system used on the web. The HTTP Basic authentication permits a client to provide credentials to an HTTP server when making a request. This method is relatively insecure, since the password is transferred in the base64 encoding, and thus can be decoded easily. The other problem is that this

method relies on simplistic credentials, namely login and password. Since we do not have in PALETTE a single identity repository, it is not feasible for the portal to provide each widget with information.

Consequently, we identified other authentication protocols, based on a shared secret between the portal and the widgets. These protocols are namely Google AuthSub (URL:AUTHSUB), AOL OpenAuth (URL:OPENAUTH), Yahoo bbAuth (URL:BBAUTH), and were consolidated in the OAuth specification (URL:OAUTH) effort. OAuth is the current state of the art protocol in API authentication, and is implemented on all web services API at Google. It provides a mean to authenticate a service *A* on a service *B* without exchanging the user credentials between them, and it involves an API key. But OAuth is mainly targeted at consumer sites that contact an identity authority, and then are redirected to the consumer site. This mechanism is called "3-legged OAuth" (URL:3LEG) and a "2-legged OAuth" was made in order to comply with the use case of OpenSocial applications (the OpenSocial specification (URL:OSOC) is based on the Google Gadgets API (URL:GADG), so it is similar in terms of functionalities), and it consists in a mean for a widget provider to make some requests to the portal in order to gather authentication informations. This mechanism is seamless in terms of user experience, as there is no need to ask for user credentials for each widget.

Finally, we wanted the widgets to declare a priori in their xml descriptors their behaviour regarding the authentication policy: if a user adds a widgets that will ask for its identity, he will be warned when adding the widget in his portal, and a visual information will remind him that he is implicitely trusting this widget. Then, each time the widget is rendered, the authentication is automatically transferred from the portal to these widgets. This approach is different from the "2-legged OAuth" approach, and it seemed to be a simpler one, explaining why we preferred it to the OAuth protocol. In future releases of the portal, it might be possible to implement an alternate authentication system based on 2-legged OAuth.

At this time, the identities of the users that are transmitted to PALETTE Services using the Widget Authentication Mechanism are represented by their unique identifier within the Portal. Because we do not have a centralized identity repository in PALETTE, the identifier of a user must be the same on the entire collection of available PALETTE Services within a CoP if we want the WAM working efficiently. Nevertheless, when the Single Sign-On using the OpenID technology (URL:OPENID) will be adopted and implemented among PALETTE partners, we may use the OpenID identifier of users as the identity data transmitted by the WAM in order to set up a more homogeneous way to transmit identity across PALETTE services.

3.4 Services Browser

As detailed in D.IMP.06, the PALETTE Service Browser (PSB) enables users to browse services in the PALETTE Services Registry Framework (PSRF) and widgets in PALETTE Service Portal (PSP). We describe here the new functionalities developed since the first version.

3.4.1 Search by categories or by tags for all services

The browser (URL:SRVB) now gives the possibility to search services by categories and by tags, as it was the case for widgets. Each category and each tag are retrieved from the PSRF: the description of all the services is analysed and lists of categories and tags are generated from this analysis. Search using tags comes to a basic keyword-based search. For each category and each tag, the number of matching services is provided on the interface. Additionally, tags including another one (according to their names) include the later matching services as results.

Figure 9: PALETTE Service Browser.

As for widgets, tags are now displayed below the title. Figure 9 shows for example the case of Amaya, where the following tags can be seen: “CSS”, “Document Authoring”, “HTML”, “MathML” and “Web”. The category (here, “information service”) is indicated under the title, at the end of the line.

The template colour has been a little changed in order to differentiate it from the CroSSE, which uses the same template.

3.4.2 Service Browser widget

A widget similar to the CroSSE’s one is proposed in PALETTE portal. This widget proposes to search services by keywords, and then shows the list of corresponding services, with the service title and a link to details. It is shown in Figure 10. Note that a click on  allows accessing the Web interface.

Figure 10: CroSSE widget.

3.5 Usage and usability of the PALETTE Services Portal and the Service Browser

While only the PSP has been experimented in a CoP, usability tests have been conducted on both the Portal and the PSB. As the development was guided by the PDM (Participatory Design Methodology), returns on experience of CoPs as well as the tests results have been taken into account to enhance the whole PALETTE Services Delivery Framework.

3.5.1 Deployment: an experimentation with a CoP

The Portal has been deployed for the CoPe-L (see URL:COPEL), with 6 widgets from 4 different services. Actually, there is one widget by service used by the CoPe-L, namely the blog, the agenda, the Contacts, and BayFac. The two other widgets concern the RSS feeds linked to the last messages posted on the blog and to the last resources added on BayFac.

The CoP's members have the choice to see each widget or not, they can adjust the portal composition according to their needs. Thus they have a global view of the services they mostly use, and a simplified access to these services.

According to feedbacks gathered during regular meetings between the CoP's members and the Portal development team, the CoP's members appreciated the fact to have a direct access to all services, and in particular to the RSS feeds. Moreover they find an added-value in customizing the portal easily, they found the service easy to tame. Compared to the tools offering the same functionalities, the members liked the personalisation of the portal to the services used by the CoPs, the possibility to add RSS feeds and also the humanization due to the widget "Contacts" which allows seeing the list of the CoP's members with their photography.

Concerning the Services Browser, it has only been tested against its usability, in the Didactic CoP, as we will see in the next section.

3.5.2 Usability tests for the Service Browser and PALETTE Services Portal

In the PALETTE project we use the usability tests as a formal approach for gathering users' feedbacks. Indeed, these tests are mainly used for measuring the quality of the tools (Web applications, desktop applications) and also the degree of satisfaction of the users. Moreover, by analysing these tests we can give recommendations to developers in order to first, adapt some functionalities of these tools and finally to produce usable tools allowing users to achieve specified goals with effectiveness, efficiency and satisfaction. Especially, two specific deliverables (D.PAR.04 and D.PAR.07) are dedicated to usability analysis. In these two deliverables the usability tests are based on the Bastien Scapin method. This is a complete and elaborated approach for testing in deep the tools' functionalities and its main objective is to guide the development of this tools. However, the usability tests of the Palette Service Portal and the Palette Services Browser is aiming at giving general recommendations because both websites are not tools with rich functionalities but it's mainly a kind of frameworks for integrating existing services: the PSP is developed for integrating developed widgets which are an aggregation of existing services and functionalities and the PSB is used for browsing existing services. Thus, we choose to use another method for usability tests of the PSP and the PSB. Indeed, in our case, we used for our tests the Computer Science Usability Questionnaire (CSUQ) developed by IBM, but many other questionnaires are reported in the literature such as SUS (System Usability Scale) and QUIS (Questionnaire for User Interface Satisfaction). We organised the tests as follows: the PSB and the PSP were initially tested respectively by the mediator and other members of Didactic and CoPeL who are the final users of these systems. Second, these users filled in the CSUQ usability test and reported more specific problems encountered during the use of these systems for accomplishing specific tasks. We will present in the following the CSUQ questionnaire.

The CSUQ Questionnaire

The CSUQ questionnaire is composed of 19 questions (see Figure 11) and each question is a statement rated on a seven-point scale from "Strongly Disagree" to "Strongly Agree". For each participant, the score of each question is calculated as follows: the scale's value of the rating given to the question is divided by the maximum score possible (7). For example, a rating of 4 given to the question1 on the

CSUQ questionnaire is giving a score of 0.57 (which represents 4 divided by 7). Moreover, 3 additional factors are defined for measuring the System Usefulness, the Information Quality and the Interface Quality as follows:

- SYSUSE (System Usefulness): question 1 through 8
- INFOQUAL (Information Quality): question 9 through 15
- INTERQUAL (Interface Quality): question 16 through 18

The value of each factor is simply the average of the score of corresponding questions. For example, the value of the SYSUSE factor is the score's average of the question 1 until the question 8.

Overall Reaction to the Website		1	2	3	4	5	6	7		NA
1. Overall, I am satisfied with how easy it is to use this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
2. It was simple to use this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
3. I can effectively complete my work using this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
4. I am able to complete my work quickly using this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
5. I am able to efficiently complete my work using this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
6. I feel comfortable using this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
7. It was easy to learn to use this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
8. I believe I became productive quickly using this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
9. The website gives error messages that clearly tell me how to fix problems	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
10. Whenever I make a mistake using the website, I recover easily and quickly	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
11. The information (such as online help, on-page messages, and other documentation) provided with this website is clear	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
12. It is easy to find the information I need	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
13. The information provided by the website is easy to understand	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
14. The information is effective n helping me complete the tasks and scenarios	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
15. The organization of information on the website pages is clear	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
16. The interface of this website is pleasant	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
17. I like using the interface of this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
18. This website has all the functions and capabilities I expect it to have	strongly disagree	○	○	○	○	○	○	○	strongly agree	○
19. Overall, I am satisfied with this website	strongly disagree	○	○	○	○	○	○	○	strongly agree	○

Figure 11: The CSUQ Questionnaire.

Usability test of the PALETTE Services Portal (PSP)

A usability study of PSP has been conducted with the CoPeL CoP. We expose in the following the results of this study. All given recommendations below are based on the scores of three factors: System Usefulness (SYSUSE), Information Quality (INFOQUAL) and Interface Quality (INTERQUAL). The usability test of the PALETTE Portal has been realised by 6 mediators of the CoP who have tested this application and filled in the CSUQ questionnaire. In Figure 12, we present the scores of the 19 items of the CSUQ questionnaire.

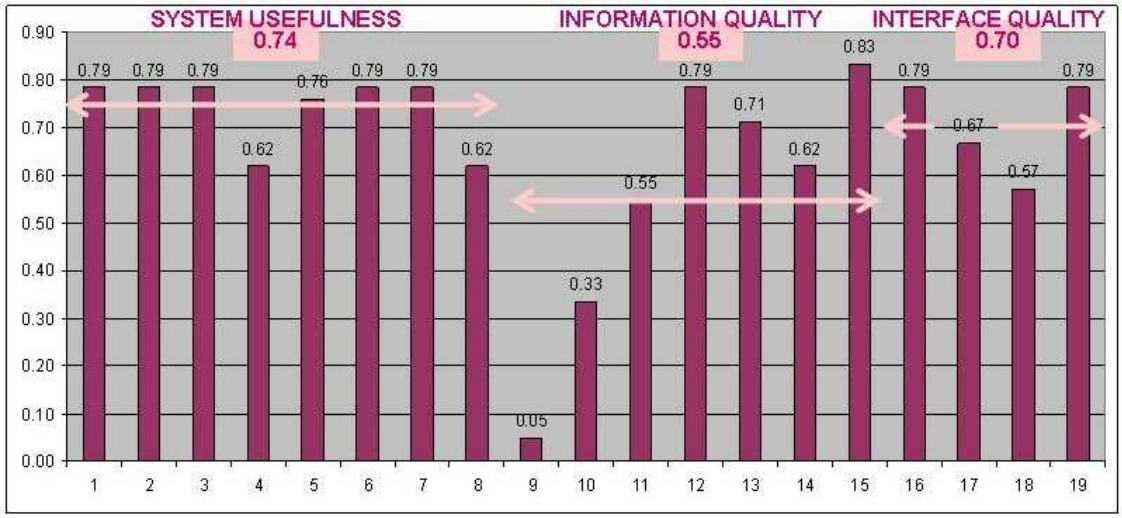

Figure 12: Results of the CSUQ for the PSP.

In general the score of the overall questionnaire is acceptable (0.6641). In addition, the score of each factor SYSUSE, INFORQUAL and INTERQUAL are as follows:

- SYSUSE (System Usefulness, Questions 1-8) = 0.7410
- INFOQUAL (Information Quality, Questions 4-15)= 0.5544
- INTERQUAL (Interface Quality, , Questions 16-19) = 0.7023

Recommendations

According to the analysis of the scores, some general recommendations have been mentioned, which have been since taken into account in the PSP:

1-According to the score of the INFOQUAL factor (0.5544), the PALETTE Portal might provide more messages guiding the users for their problem recovery and also for helping them using the PSP.

2-The score of the INTERQUAL factor (0.7023) is acceptable. However, the users aren't satisfied of the functions they suspect to have in the PSP. Thus, the PP might integrate the option of browsing the existing widgets (like in the PALETTE Service Browser) and to add them easily to the PSP.

Usability test of the PALETTE Service Browser (PSB)

Like was done for the PSP, usability tests have been conducted on the PSB, based on the CSUQ questionnaire. Those tests have been performed with the Didactic[1] CoP. The usability test of the PALETTE Service Browser has been performed by 5 users (mediators and other members of the CoP) who have tested this application and filled in the CSUQ questionnaire.

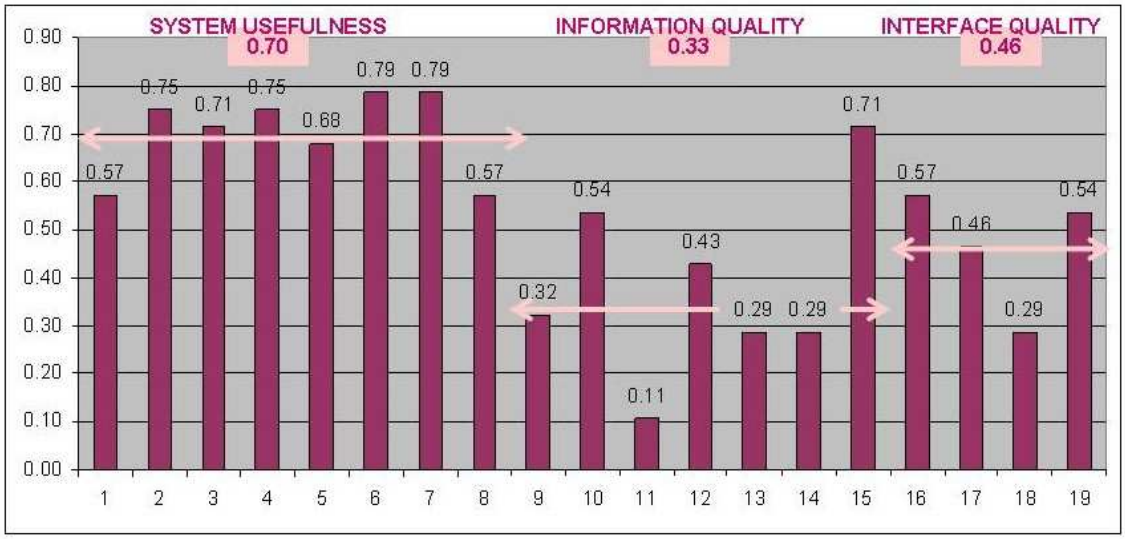

Figure 13: Results of CSUQ for the PSB.

The whole score of the questionnaire is equal to 0.53, which is acceptable. In addition, the score of the 3 factors SYSUSE, INFORQUAL and INTERQUAL are as follow:

- SYSUSE (System Usefulness, Questions 1-8) = 0.70
- INFOQUAL (Information Quality, Questions 4-15) = 0.33
- INTERQUAL (Interface Quality, Questions 16-19) = 0.46

Recommendations

Following the different scores obtained, two recommendations have been proposed, which have been since integrated in the PSB:

1.Concerning the score of the INFOQUAL factor (0.33), the PSB might provide more explicit information and propose a help module. More specific recommendations are mentioned in the following:

- Add description about main objective of the Website in the home page
- Add a help module explaining the whole objective and main functionalities of the PSB
- Put the description of each service close to its screenshot
- The PSB might provide more indications for problems recovery: the description of the error and how he can resolve it

2.Concerning the score of the INTERQUAL factor (0.46), two general recommendations can be mentioned in the following:

- Caption of buttons and text might be more explicit to avoid confusion for user

[1] http://palette.ercim.org/content/view/40/37/

- Create a button (which is different from an icon) to indicate how to add tags

4 Conclusion

Together with the offered PALETTE of services, the PALETTE project provides two frameworks for helping communities of practices to get the most of their IT environment. We have presented in this document the last versions of the framework for services registration, the PSRF, and of the framework for services delivery to the end-user, the PSDF. The enhancements since the first versions presented in previous deliverables, have been detailed and a usability study has been performed and reported. To conclude with this work, we discuss here the main achievements and perspectives of development, as well as the usability study's results.

Concerning the PALETTE Registry, the final version will be the central registration point for all the available PALETTE tools and services. After the finalization of the PSRF development, the focus is currently turned on the assistance of the service registration procedure. Through the PALETTE Registry, UI partners will have the ability to manage their published services and to modify the provided service descriptions in a very easy way. Thus, all the PALETTE partners may have valuable feedback from the exploitation of the registry through the delivery framework.

This report also details the introduction of Widget Events into the portal, and the corresponding API that has been developed and integrated in the current release of the portal. This is a significant step towards inter-widget communication and towards the introduction of the Drag and Drop interaction technique into Web based portals. Until now, that interaction technique which is at the heart of the direct manipulation style of most Desktop environment was not available on the Web. Those new features now allow widget developers to create interactive widget aggregates able to propose more synergistic use cases and reinforce collaboration between PALETTE services at the portal level. In particular the provision of such mechanisms is a necessary step to allow widget developers to create potential chains of interactions between several PALETTE services, as it has been outlined in the generic and instantiated scenarios. Some examples of such interactions should be ready for experimentation soon, and they will be described in the next deliverable, D.IMP.08.Some usability issues are still under consideration to find the best way to provide the feedbacks required for a drag and drop operation. Finally, regarding the inter-widget communication, it could be interesting to provide some means for users to have visual feedbacks on event propagation, and to be able to control the event flow, for example by decoupling already coupled widgets.

The Widget Authentication Mechanism solves the problem relevant to automatic authentication of users, which was raised during the elaboration of the generic and instantiated scenarios. It solves it by accessing services through widgets, and by providing a secure identity transportation system between the portal and compliant PALETTE services. Despite the fact that we have taken care in designing a secure mechanism, it is difficult to guarantee that it is flawless. If the portal is used to display critical information, it could be useful to realize a security audit on this software.

The usability tests for the portal (PSP) and the services browser (PSB) revealed an efficient approach to measure the degree of satisfaction of the CoP mediators and also the quality of these tools. Globally, the users gave a good feedback about both applications, but still proposed some recommendations. For the PSB, the CSUQ analysis revealed two concerns: information's quality and interface's quality. For the PSP, only information's quality was detected as a concern. The provided recommendations were mainly integrated by these tools' developers in order to accommodate the expectations of CoP mediators. Because usability was taken into account at the end of the development process, it was used as a validation of the work that has been done on user interfaces. Some problems were found and fixed in the two audited softwares, following the recommendations arisen from the usability study. These tests are interesting since they truly represent the point of view of the future users of these services. However, it would have been interesting to integrate them in the whole development process, following for example the ISO 13407 approach. Another interesting approach could be to compare the results coming from the user tests and experts' tests, done on the basis of criteria like Bastien-Scapin ones, in order to further improve usability.

5 References

5.1 Bibliography

Connolly D., Ed. (2007). Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation 11 September 2007, available at www.w3.org/TR/grddl/

Hepper S. (2006). Java™ Portlet Specification Version 2.0 Early Draft 1, available at http://jcp.org/en/jsr/detail?id=286

5.2 Webography

All the Web references have been accessed in August 2008.

3LEG The 2-legged and 3-legged OAuth variants, at http://feeds.feedburner.com/RecentlyAddedFreeIphoneApplications-PinchMedia

APR AES Proposal: Rijndael, at http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

AUTHSUB The Google AuthSub authentication protocol, at http://code.google.com/apis/accounts/docs/AuthSub.html

BBAUTH The Yahoo bbAuth authentication protocol, at http://developer.yahoo.com/auth/

BGF Better Gmail 2 Firefox Extension for New Gmail, at lifehacker.com/software/exclusive-lifehacker-download/better-gmail-2-firefox-extension-for-new-gmail-320618.php

COPEL `http://palette.ercim.org/content/view/144/37/`

CRWE The CroSSE Web interface, at http://sim.tudor.lu/PALETTE/CroSSE/

CRWI The CroSSE widget, at http://sim.tudor.lu/PALETTE-demo/portal/

CRWS The CroSSE webservice, at http://sim.tudor.lu/PALETTE/CroSSE/webservice.php?keywords=

GADG Google Gadgets API, at http://code.google.com/apis/gadgets/docs/dev_guide.html¨

HTML5 HTML 5 Draft Recommendation - 4 September 2008, at http://www.whatwg.org/specs/web-apps/current-work/#crossDocumentMessages

MIM List of existing mime-types, at http://www.mimetype.org/

OAUTH The OAuth authentication protocol, at http://oauth.net/

OPENAUTH The AOL OpenAuth authentication protocol, at http://dev.aol.com/api/openauth

OPENID The OpenID specifications, at http://openid.net/developers/specs/

OSOC The OpenSocial API, at http://code.google.com/apis/opensocial/docs/0.8/spec.html

PEAR The PHP Extension and Application Repository, at http://pear.php.net/

PHPD phpDocumentor: The complete documentation solution for PHP, at http://www.phpdoc.org/

PSRF The PALETTE Services Registry Framework at http://150.140.18.39:91/psrf/

PSRF-SERV PALETTE Services Registry Framework service page, at http://PALETTEregistry.cti.gr

RFC2617 RFC 2617, at http://tools.ietf.org/html/rfc2617

SRVB The Service Browser at http://sim.tudor.lu/PALETTE-demo/srvBrowser/

TLOI	The Laws of Identity at, http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf
UWA	Universal Widget API, at http://dev.netvibes.com/doc/uwa_monopage
WRWD	W3C Widgets 1.0 Requirements working draft, at http://www.w3.org/TR/widgets-reqs/

6 Annexes

6.1 PALETTE Widget format specification

6.1.1 Introduction

Widgets, as they are used within the PALETTE portal, are small Web applications for displaying and updating remote data and serve as a common entry point to the end user. The PALETTE Widget format is inspired by the W3C Widgets 1.0 Draft (URL: W10) with small implementation-specific adaptations and extensions, while being fully compatible to the original Widgets 1.0 specification.

The PALETTE widgets specification defines two different types of widgets: local and remote widgets. A local widget is a bundled archive of files that is deployed within the portal. Remote widgets are stored on a remote server and only registered within the portal by supplying their manifest, which contains the URI where the widget can be found. They can be written in any language as long as they generate valid HTML. Differences between local and remote widgets exist in the packaging, some elements within the manifest *config.xml* as well as the Widget Scripting Interfaces.

6.1.2 Widget Packaging

File format

The file format only applies to local widgets, as they are uploaded and deployed within the portal. Remote widgets can use whatever file format their implementation language requires, as long as they can be referenced by an URI.

Local widgets are a bundled archive of files, as specified by the Widgets 1.0 Draft File Format (URL: W10F).

6.1.3 Widget files

Every widget *must* define the following two files:

The *config.xml* file:

This manifest file contains information necessary to initialise the widget. It always contains information about the widget's name, identity and geometry and may optionally contain more information about the widget, such as the widget description, author information and an icon reference.

An index file:

This is the main document of the widget, which is displayed in the portal and whose main properties are established by the *config.xml* file. For local widgets, the index file *must* be called *index.html*. For remote widgets, this *must* be a valid index file. Since remote widgets are hosted on remote servers, the validity of the index file name depends on the web server configuration. The index document can reference external content and *should* produce valid HTML or XHTML markup.

Widget Folder Structure

The widget folder structure only applies to local widgets. Remote widgets can use any folder structure as long as the widget can be referenced by an URI pointing to its folder.

The *config.xml* and *index.html must* be at the root of the .zip file, with any associated resources, such as scripts and images, in the same directory or subdirectories.

6.1.4 Widget Configuration File: config.xml

Necessary information to run and display a widget within the portal are stored in a file named *config.xml*. The *config.xml* file is an XML document.

Since the PALETTE widget specification is an extension of the W3C Widgets 1.0 specification, two different namespaces are used to distinguish between the two types of elements. These namespaces have to be defined in the root *widget* element: the default W3C namespace *must* be *http://www.w3.org/TR/widgets/* and the PALETTE namespace *must* be *http://PALETTE.ercim.org/ns/*

A minimal *config.xml* looks like the following, giving the widget a name, an initial viewport of 300×300 pixels and an id:

```
<widget xmlns="http://www.w3.org/TR/widgets/"
xmlns:PALETTE=http://PALETTE.ercim.org/ns/
        id=”helloWorldWidget”
        width=”300”
        height=”300”>
        <title>Hello World!</title>
</widget>
```

For the *config.xml* file, the same structural restrictions apply as for the Widgets 1.0 Draft, specified in chapter 3 (URL: W10C).

To validate the *config.xml* file, two XML Schema files are provided with this document.

The widget Element

The *widget* element is the root element of the *config.xml* file and *must* be present. It *should* contain, in any order, exactly one title, and optionally one of each author, description and icon.

Please note that the widget id attribute is mandatory (contrary to Widgets 1.0 Draft) and is used to identify the widget internally in the PALETTE portal.

The id attribute

The *id* attribute *must* be present in *config.xml* and bound to the *widget* element. The identifier should be a valid XML Schema ID (URL: XSDID).

The width and height attributes

The *width* and *height* attributes *must* be present in *config.xml* as children of the *widget* element. After stripping of any leading/trailing white space, the value of this element *must* be interpretable as a string representation of an integer, containing only the characters [0-9].

Please note that, while these integers give the initial width and height of the viewport of the widget, measured in CSS pixels (see section 4.3.2 of URL: CSS), only the *height* integer is directly respected by the portal, while the width of the widget is defined by the portal layout. The *width* integer serves only for compatibility with the W3C specification and might be used in the future to display the widget as a Desktop widget (URL: WD).

The title Element

The *title* element *must* be present in *config.xml* as a child of the *widget* element. It *should* contain a string whose purpose is to provide a human-readable title for the widget that can be used for example in application menus and similar contexts.

The author, description and icon Elements

These elements respect the specification given by the Widgets 1.0 Draft. Please refer to chapter 3 of the Widgets 1.0 Draft for further details.

The widget_type Element

The *widget_type* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

The *widget_type* is an optional element of the *widget* element and the absence of the *widget_type* element means that the widget is a local widget. It *should* contain a string whose only allowed values are either *local* or *remote*.

```
<PALETTE:widget_type>local</PALETTE:widget_type>
```

The widget_location Element

The *widget_location* element applies only to remote widgets. It has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

If the *widget_type* element value is *remote*, the *widget_location* element *must* be present in *config.xml* as a child of the *widget* element. If the *widget_type* element value is *local*, this element is ignored. The value of this element is interpreted as a syntactically valid URI pointing to the folder containing the remote widget index file. If no trailing slash is present, the portal will automatically add one.

```
<PALETTE:widget_location>http://path.to/my/remote/widget/</PALETTE:widget_locati
```

The alternate_url Element

The *alternate_url* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

The *alternate_url* is an optional element of the *widget* element and allows to specify an alternate view of the resource, the widget is about. In the case of remote widgets, *alternate_url* can be used to provide an URL to the entire application of which the widget shows only one functionality.

The preferences Element

The *preferences* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

The *preferences* element allows to specify customisable user preferences that are stored in the PALETTE Web portal and exposed to the widgets through the Widget Scripting Interfaces. The *preferences* element contains any number of *preference* elements.

The preference Element

The *preference* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

If present, this element *must* be a child of the *preferences* element. The following table lists the *preference* attributes:

name	Required "symbolic" name of the user preference; displayed to the user during editing if no display_name is defined. Must contain only letters, number and underscores, *i.e.* the regular expression ^[a-zA-Z0-9_]+$ must be unique.

display_name	Optional string to display alongside the user preferences in the edit window. Must be unique.
datatype	Optional string that indicates the data type of this attribute. Can be string, bool, number, hidden or enumeration. The default is string.
default_value	Optional string that indicates a user preference's default value.

```
<PALETTE:preferences>
        <PALETTE:preference name="name" display_name="First name" datatype="string"/>
        <PALETTE:preference name="age" display_name="Age" datatype="number"/>
</PALETTE:preferences>
```

The enumeration Element

The *enumeration* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the PALETTE namespace.

If present, this element *must* be a child of the *preference* element. This element is ignored if the attribute *datatype* of the parent *preference* element is not of value *enumeration*. The following table lists the *enumeration* attributes:

value	Required "symbolic" value of the user preference; displayed to the user during editing if no display_value is defined. Must contain only letters, number and underscores, *i.e.* the regular expression ^[a-zA-Z0-9_]+$ must be unique.
display_value	Optional string to display alongside the user preferences in the edit window. Must be unique.

The *enumeration* data type is presented in the user interface as a menu of choices. The content of the menu is specified using the enumeration elements.

```
<PALETTE:preferences>
        <PALETTE:preference name="lang" display_name="Language" datatype="enumeration" default_value="fr">
                <PALETTE:enumeration value="fr" display_value="French"/>
                <PALETTE:enumeration value="de" display_value="German"/>
                <PALETTE:enumeration value="en" display_value="English"/>
        </PALETTE:preference>
```

The authentication Element

The *authentication* element has been added to the Widgets 1.0 Working Draft and *should* be specified in the PALETTE namespace, but is not mandatory. If present, this element *must* a child of the *widget* element. The value of the *authentication* element must be a string with a value of *'enabled'* or *'disabled'*.

```
<PALETTE:authentication>enabled</PALETTE:authentication>
```

If the *authentication* element is not set in the *config.xml* file, the default behaviour is to set the widget authentication value to *'disabled'*.

6.1.5 Widget Scripting Interfaces

The widget Object for Local Widgets

The purpose of the *widget* object is to expose functionality to widgets that are not available outside of the PALETTE portal. The *widget* object is accessible through JavaScript, but only for local widgets.

method <datatype> *preferenceForKey* (string *key*)

The *preferenceForKey*() method takes a string argument, *key*. When called, this method *shall* return the value of the user preference whose attribute *name* corresponds to the *key* provided as argument, or null if the *key* does not exist or hasn't been specified by the user.

method void *setPreferenceForKey* (<datatype> *preference*, <string> *key*)

The *setPreferenceForKey*() method takes two arguments, *preference* and *key*. When called, this method shall store the value of *preference* in the preference which attribute *name* corresponds to the *key* provided as argument. If no corresponding preference has been found (the *key* doesn't correspond to a preference specified in the manifest *config.xml*), this instruction is ignored.

method void httpG*et* (string *URI*, map *params*, function *callback*)

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP GET request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either responseXML or responseText.

method void *httpPost* (string *URI*, map *params*, function *callback*)

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP POST request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either responseXML or responseText.

method void *httpPut* (string *URI*, map *params*, function *callback*)

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP PUT request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either responseXML or responseText.

method void *httpDelete*(string *URI*, map *params*, function *callback*)

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP DELETE request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either responseXML or responseText.

method void httpG*etJSON* (string *URI*, map *params*, function *callback*)

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP GET request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is parsed as JSON into an JavaScript object.

The ***httpPost***, ***httpGet***, ***httpPut***, ***httpDelete*** *and* ***httpGetJSON*** methods above profit from a proxy installed within the PALETTE Web portal and are able to bypass the security restrictions imposed by the XMLHttpRequest object. This makes it possible to send a request to an URI that is not on the PALETTE Web server but can be any remote server.

method void *setContentProxy* (string *path*)

The *setContentProxy* method is used to specify the location of the content proxy PHP file to be used by the local development environment. The local development environment is a special environment designed to facilitate widget development without the need to deploy and test the widget within the PALETTE Web portal. It is possible to use this environment by simply including two specific files, a CSS and a JavaScript file, in the HTML code of the *index.html* file, as explained further in the Widget Tutorial. Since the proxy needs to be installed on the server the widget is executed on, it is necessary to specify the path to the PHP proxy on the local machine.

method void *setHttpCredentials* (string *username*, string *password*)

The *setHttpCredentials* method is used to specify the username and password for basic HTTP access authentication.

method void *fireWidgetEvent (*string *target*, string *eventType*, Object *data)*

The *fireWidgetEvent* method fires an event of a given *eventType* in order to transmit some *data* in the transmitted message, for a particular *target* widget alphanumeric identifier. If the *target* is set to *null*, the event will be notified to every widget instance listening for this event type.

method void *addWidgetEventListener (*string *eventType*, function *eventHandler*, Array *acceptedSources)*

The addWidgetEventListener method enables your widgets to listen to *eventType* events that are coming from a specified alphanumeric widget identifier list defining the *acceptedSources*. When events are fired, the function defined by *eventHandler* is executed. If the *acceptedSources* list is set to null, the widget will accept be notified of this type of event from any source widget.

The function *eventHandler* must accept an object as its first parameter. Indeed, this object will represent the message transmitted when events are notified to listeners. The object will be an Object instance with two properties:

- *eventType*: a string containing a URI that represents the type of event that was fired.
- *data*: some arbitrary useful to handle the event. The format of the transmitted data may differ between event types.

method void *removeWidgetEventListener (*string *eventType,* string *source)*

Stop to listen to a given *eventType* for a particular *source* widget identified by an alphanumeric widget identifier. If the **source** is set to *null*, the widget will completely stop to listen to this kind of events.

method void *bindWidgetToDropType (*string *eventType)*

Declares that a widget is a drop target for drag and drop events of a given *eventType*. This will inform the drag and drop widget engine to display a proper feedback when dragging some compatible data over the widget frame.

method void *addDragData (*Element *domElement, string eventType, Object data, string tooltip)*

Transforms a *domElement* of the widget tree into a draggable element that triggers a drag event of a given *eventType* with an associated *data* and that displays a *tooltip* message when dragging.

The widgets that registered for the drag event of type *eventType* with the *bindWidgetToDropType* method will be notified if the user drops the widget element onto them.

The onLoad() function for Local Widgets

The *onload*() function replaces the Intrinsic event *onload* specified by the HTML 4 Scripts document (URL: H4S). This event occurs when the user agent finishes loading a window or all frames within a frameset and is commonly used to execute JavaScript as soon as the page has finished loading. Since the PALETTE portal needs to initialize the *widget* object first, we discourage the usage of the onload event, since we cannot guarantee that the *widget* object has been fully loaded. The onload event should be replaced by the *onLoad*() function, if the widget needs to execute a script when it has finished loading.

The *onLoad*() function is optional and if present, is called by the PALETTE portal as soon as the widget has been fully initialised. To guarantee that the HTML DOM is ready and the *widget* object accessible, the widget logic should be executed from within the *onLoad*() function.

```
<script type="text/javascript">
        function onLoad()
        {
                /* put your widget logic here */
        }
```

6.1.6 User Preferences Access for Remote Widgets

Remote widgets access the user preferences through the GET parameters specified in the widget URL. For each user preference, for which a value has been specified or a default value is known, a (key, value) pair is generated and appended to the URL.

Apart from the widget edit window in the PALETTE portal, remote widgets have no possibility to change the value of the user preference through the Widget Scripting Interface.

6.2 Widget descriptor schema

6.2.1 Manifest.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
      targetNamespace="http://www.w3.org/TR/widgets/"
      xmlns="http://www.w3.org/TR/widgets/"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:PALETTE="http://PALETTE.ercim.org/ns/"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">

<xs:import                          namespace="http://PALETTE.ercim.org/ns/"
schemaLocation="palette.xsd"/>

<xs:element name="widget">
  <xs:complexType>
    <xs:all>
      <xs:element ref="title" maxOccurs="1"/>
      <xs:element ref="description" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="icon" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="access" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="author" minOccurs="0"/>
      <xs:element ref="license" minOccurs="0"/>
      <xs:element ref="PALETTE:widget_type" minOccurs="0" />
      <xs:element ref="PALETTE:widget_location" minOccurs="0"/>
      <xs:element ref="PALETTE:alternate_url" minOccurs="0"/>
      <xs:element ref="PALETTE:preferences" minOccurs="0"/>
    </xs:all>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="version" type="xs:string" use="optional"/>
    <xs:attribute name="height" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="width" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="start" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="title" type="xs:string"/>
```

```
<xs:element name="description" type="xs:string"/>

<xs:element name="icon">
  <xs:complexType>
      <xs:attribute name="src" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>

<xs:element name="access">
  <xs:complexType>
      <xs:attribute name="network" type="xs:boolean"/>
      <xs:attribute name="plugins" type="xs:boolean"/>
  </xs:complexType>
</xs:element>

<xs:element name="author">
  <xs:complexType>
      <xs:simpleContent>
            <xs:extension base="xs:string">
                  <xs:attribute name="url" type="xs:anyURI"/>
                  <xs:attribute name="email" type="xs:string"/>
            </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="license">
  <xs:complexType>
      <xs:simpleContent>
            <xs:extension base="xs:string">
                  <xs:attribute name="href" type="xs:anyURI"/>
            </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
</xs:element>

</xs:schema>
```

6.2.2 Palette.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema                      xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://PALETTE.ercim.org/ns/"
xmlns="http://PALETTE.ercim.org/ns/">

<xs:element name="widget_type">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="local"/>
      <xs:enumeration value="remote"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="widget_location" type="xs:anyURI"/>

<xs:element name="alternate_url" type="xs:anyURI"/>

<xs:element name="preferences">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="preference"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="preference">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="enumeration"/>
    </xs:choice>
    <xs:attribute name="name" type="identifier" use="required"/>
    <xs:attribute name="display_name" type="xs:string" use="optional"/>
    <xs:attribute name="datatype" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="string" />
```

```
          <xs:enumeration value="bool" />
          <xs:enumeration value="number" />
          <xs:enumeration value="hidden" />
          <xs:enumeration value="enumeration" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="default_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="enumeration">
  <xs:complexType>
    <xs:attribute name="value" type="identifier" use="required"/>
    <xs:attribute name="display_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="identifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9_]+"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```

6.3 PALETTE Widgets 1.0 Tutorial

6.3.1 Introduction

The PALETTE Widgets 1.0 specification has been designed with the clear goal in mind to make the creation of PALETTE Widgets as simple as possible. Only a few simple steps are necessary to make a Widget compatible with the requirements, after which you can entirely focus on the content of your Widget. This tutorial will guide you through the creation of a manifest *config.xml* and explain the few guidelines you need to follow to develop a PALETTE Widget, either local or remote.

6.3.2 Widget Configuration File: config.xml

Basic Configuration File Structure

Within a new directory, create a new text file called *config.xml*. This file contains all the information the PALETTE portal needs to deploy and run your widget as well as to save the user preferences

within the portal. As you can see in the PALETTE Widgets 1.0 specification, only a few elements are mandatory and a minimal *config.xml* looks like the following:

```
<widget widget xmlns="http://www.w3.org/TR/widgets/"
   xmlns:PALETTE=http://PALETTE.ercim.org/ns/
   id="helloWorldWidget"
   height="300"
   width="300">
   <title>Local Hello World</title>
</widget>
```

If you are using an XML editor capable of validating an XML file against an XML Schema, you can modify the *widget* element to include the location of your schema:

```
<widget xmlns=http://www.w3.org/TR/widgets/
   xmlns:PALETTE="http://PALETTE.ercim.org/ns/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
   id="helloWorldWidget"
   height="300"
   width="300">
   <title>Local Hello World</title>
</widget>
```

For the XML Schema validation to work, both the manifest.xsd and the palette.xsd should be in the same directory as the config.xml file. Using all of the elements defined by the Widgets 1.0 Draft, a more complete manifest could look like the following:

```
<widget xmlns="http://www.w3.org/TR/widgets/"
   xmlns:PALETTE="http://PALETTE.ercim.org/ns/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
   id="helloWorldWidget"
   height="300"
   width="300">
   <title>Local Hello World</title>
   <author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent
Haan</author>
   <description>A simple Hello World widget</description>
   <icon src="images/icon.gif"/>
</widget>
```

PALETTE Specific Configurations

In order to use any of the PALETTE Web portal features, we have the possibility to use the elements declared in the PALETTE namespace, which allow us to define the type of widget we're going to create and the user settings the portal should store.
To have the PALETTE Services Portal store the name of the widget user, we first need to declare a preference:

```
<PALETTE:preferences>
   <PALETTE:preference name="username" display_name="Username"
datatype="string" default_value="guest"/>
</PALETTE:preferences>
```

The complete config.xml file would look like the following:

```
<widget xmlns="http://www.w3.org/TR/widgets/"
   xmlns:PALETTE="http://PALETTE.ercim.org/ns/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
   id="helloWorldWidget"
   height="300"
   width="300">
   <title>Local Hello World</title>
   <author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent
Haan</author>
   <description>A simple Hello World widget</description>
   <icon src="images/icon.gif"/>
   <PALETTE:preferences>
     <PALETTE:preference name="username" display_name="Username"
datatype="string" default_value="guest"/>
   </PALETTE:preferences>
</widget>
```

Widget Authentication Mechanism

Another PALETTE widget feature is to enable widgets to automatically authenticate themselves and their users against PALETTE Services. This feature is named “Widget Authentication Mechanism”, and enables the portal to send request on behalf of the widget.

The “Widget Authentication Mechanism” solves the problem relevant to automatic authentication of users through widgets by providing a secure identity transportation system between the portal and compliant PALETTE services.

When enabled, the portal’s proxy will inject on the fly some encrypted information about the widget and the user using it in requests’ headers each time a request to the relevant PALETTE Service is performed.

By decrypting the data, the PALETTE Service will be able to authenticate both the widget’s user and the widget by using a symmetric encryption key shared by the portal and itself.

To enable the “Widget Authentication Mechanism” for your widget, simply add the following line into your *config.xml*, between the *widget* element tags.

```
<PALETTE:authentication>enabled</PALETTE:authentication>
```

If you do not want to use this feature for your widget, simply omit the *authentication* element or declare it unequivocally by using the following example.

```
<PALETTE:authentication>disabled</PALETTE:authentication>
```

Widget scrollbar

You may want to enable vertical scrolling for your widget if its height may vary during execution. To do so, simply add the following line into your *config.xml*, between *widget* element tags.

```
<PALETTE:scrollable>true</PALETTE:scrollable>
```

If you do not want to enable vertical scrolling for your widget, omit the *scrollable* element or declare it by inserting the following line your *config.xml* file.

```
<PALETTE:scrollable>false</PALETTE:scrollable>
```

6.3.3 Widget Index File

Creating a Local Widget

The local widget is the default type of PALETTE Widget. It needs to be packaged and deployed in the PALETTE Web portal but has less technical restrictions because the Widget is located locally on the server. In the case of a local widget, this file must be called index.html and the HTML code for our "Local Hello World" Widget looks like the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>
</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
</HTML>
```

To access the user preferences, we use the Widget Scripting Interface, more specifically, the *widget* object that is only available within the PALETTE Web portal. Since this object might not be available immediately after the onload event, we strongly recommend to use the onLoad() function as explained in the PALETTE Widget 1.0 specification. If present, this function will be called by the PALETTE Web portal as soon as the entire object has been loaded and should replace the default onload event for local widgets.
The complete JavaScript code that replaces the username with the value within the widget object looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>

<script type="text/javascript">
   function onLoad()
   {
      document.getElementById('username').firstChild.nodeValue =
widget.preferenceForKey('username');
```

```
   }
</script>

</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
</HTML>
```

Please note that the argument given to preferenceForKey() is the value given for the *name* attribute for our preference in the config.xml file.

Testing a Local Widget

To facilitate development of local widgets, a special development environment is available that simulates the PALETTE Web portal. This testing environment is very easy to use and only requires two lines to be added to the local widget index.html file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>

<link rel="stylesheet" type="text/css"
href="http://www.PALETTE.tudor.lu/widget/css/PALETTE.css" />
<script type="text/javascript"
src="http://www.PALETTE.tudor.lu/widget/js/PALETTE.js"></script
>

<script type="text/javascript">
   function onLoad()
   {
      document.getElementById('username').firstChild.nodeValue =
widget.preferenceForKey('username');
   }
</script>
</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
</HTML>
```

These two lines will load the most recent version of the development environment and display the local widget in much the same way it would appear within the PALETTE Web portal. The JavaScript file includes the config.xml parser that additionally checks your config.xml for syntax errors and the *widget* object that would only be available within the portal. Since no user settings are available for the development environment, the *default_value* of the preference will be used instead.

Testing a Local Widget with the Content Proxy

The content proxy is a script (a PHP file) that transparently loads web pages from remote servers, without the XMLHttpRequest security restrictions. It is installed within the PALETTE Web portal and allows widgets to access content from remote servers. Since the local development environment has no such proxy, we provide you with the necessary functionalities to install and use a proxy on your local machine.

The first necessary step is to download and copy the contentProxy.php file to your local machine and make it accessible by your local web server. Since the script is a PHP file, you need a webserver capable of interpreting PHP files as well. Please notice that the PHP proxy requires the CURL package to be installed and properly configured.

Secondly, you should use the setContentProxy method of the widget object to set the path to the proxy file. Afterwards, you can start using the get and post methods of the widget object as usual.

An example of a widget using a local content proxy to read the content of a text file can be found below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Widget Object Test</title>
<link rel="stylesheet" type="text/css" href="http://www.PALETTE.tudor.lu/widget/css/PALETTE.css" />
<script type="text/javascript" src="http://localhost/widget/js/PALETTE.js"></script>
<script type="text/javascript">
function onLoad(){

 widget.setContentProxy('./contentProxy.php');

 widget.httpGet('a.txt', null, function(data){
  document.getElementById('user').appendChild(document.createTextNode(data));
 });
}
</script>
</head>
<body>
Bonjour <span id="user"></span>.
</body>
</html>
```

Creating a Remote Widget

Contrary to local widgets, remote widgets are hosted on a remote server. They can be written using any available technology, as long as the output is valid HTML or XHTML and they are valid index files on the remote server. In case of remote widgets, the user preferences are sent by GET parameters, which doesn't require any particular development environment to simulate.

Before we are able to create a remote widget, we first need to modify the config.xml file accordingly:

```
<widget xmlns="http://www.w3.org/TR/widgets/"
   < xmlns:PALETTE="http://PALETTE.ercim.org/ns/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
```

```
    id="helloWorldWidget"
    height="300"
    width="300">
    <title>Remote Hello World</title>
    <author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent
Haan</author>
    <description>A simple Hello World widget</description>
    <icon src="images/icon.gif"/>
    <PALETTE:widget_type>remote</PALETTE:widget_type>

<PALETTE:widget_location>http://path.to/my/widget/folder</PALE
TTE:widget_location>
    <PALETTE:preferences>
       <PALETTE:preference name="username" display_name="Username"
datatype="string" default_value="guest"/>
    </PALETTE:preferences>
</widget>
```

If we want to create a remote Hello World widget in PHP, all that is left to do is create an "index.php" file at the address specified by the *widget_location* element and write the corresponding PHP code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>
</HEAD>
<body>
Hello <span id="username"><?php echo $_GET['username']; ?></span>!
</body>
</HTML>
```

7 Table of figures