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.06 – First version of PALETTE services delivery framework

Due date of deliverable: March 31, 2008

Actual submission date: April 22, 2008

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

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

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

Keyword List: service delivery, framework, user interface, widget, portal, mashup

Responsible Partner: CRP-HT

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
0.1	23-01-2008	Draft	JDL
0.2	27-02-2008	Draft	JDL
0.3	04-03-2008	Draft	MLW
0.4	05-03-2008	Draft	MLW
0.5	11-03-2008	Draft	YNA: homogenisation, corrections and notes, + inclusion of SSI part on the overview for Palette services integration in the presentation layer
0.6	11-03-2008	Draft	SSI: cleaning up Service Browser section, updated references and webography
0.7	13-03-2008	Draft	YNA: Added part of CTI on PSRF API, added updated part of SSI and AVA's part on Palette service integration in the presentation layer
1.0	18-03-2008	Ready for submission to reviewers	YNA: Summary, conclusion, faults corrections, homogenisation of the sections and figures, reordering of acronyms and webography
1.1	11-04-2008	Release candidate	YNA: changes according to reviewers remarks
1.2	22-04-2008	Ready for SC validation	YNA

Deliverable manager

- Yannick Naudet, CRP-HT

List of Contributors

- Jean-David Labails, Yannick Naudet, Alain Vagner, Marie-Laure Watrinet, CRP-HT
- Stéphane Sire, EPFL
- George Gkotsis, Manolis Tzagarakis, CTI

List of Evaluators

- Nikos Karousos, CTI
- Amaury Daele, UNIFR

Summary

This document presents the first version of the Palette Services Delivery Framework (PSDF), which provides the end-user tools to search, choose and exploit Palette services. Search and choose can be performed using a web access: the Palette Service Browser, which provides access to registered services and widgets thanks to the PSRF API. Exploitation is in particular provided by the Palette Service Portal, which is a customizable portal based on the W3C Widgets proposal, allowing end-users to access services through small web applications (widgets). An overview of methods to integrate Palette services at the presentation layer is provided. Finally, a solution for global-awareness between services and user is presented: the Cross Awareness Knowledge Base (CAKB). Annexes are included, explaining in particular how to create widgets and integrate them in a portal, and providing the CAKB ontology and example of RSS feed it uses.

1 Introduction

This document presents the first version of the Palette Services Delivery Framework (PSDF) and the elements constituting it. The PSDF is dedicated to end-users, and thus CoPs, providing them with means to efficiently find services registered in the PSRF, integrate them either in existing architectures or in a dedicated personalisable user interface, and finally monitoring actions performed on services at runtime.

The tools constituting the PSDF provide solutions answering main concerns of both CoPs' members as end-users of PALETTE services and services providers. From a CoP's member perpective those concerns are the following:

- Which are the available services? How to find them and use them?
- How to use all of the services from one central place?
- How to have a global view of knowledge and activities in the CoP?

For the first point, we provide the Palette Services Browser (PSB), a web portal that will allow end-users to browse, choose, install and use PALETTE services that providers have made available. For the second point, we provide the Palette Services Portal (PSP), a web application that allows CoPs' members to organise their own views and interfaces to deployed PALETTE services using small-embedded applications called widgets. Finally, the Cross-Awareness Knowledge Base (CAKB), which provides CoPs with awareness concerning what happends in the different used Palette services, provides an answer to the third point.

From a service provider, we can list those concerns:

- How to make my services available to a CoP's members?
- How to make my service useable with the other services from a central place?
- How can my service diffuse knowledge, activities and related actions it manages to other services?

Services providers can cope with the first point by describing their services according to the PALETTE services description model presented in D.IMP.04 and upload it into the PSRF. The second point can be achieved with the building of widgets that can be used in the PSP. Finally, the third point can be solved if providers implement RSS feeds for the CAKB.

The remainder of this document details the tools of the PSDF, which we propose to answer the above listed concerns. We present in section 2 the Palette Services Browser. Then, section 2.2 describes the Palette Services Portal. The specifications are given, underlying technologies are described, and tutorials are provided to help developers making widgets for their services, and users to create their personel portal. The section 4 is dedicated to the Cross-Awareness Knowledge Base. Finally, the last section provides conclusions and perspectives of future work, remaining open issues to be solved for next releases.

1.1 Acronyms

- API : Application Programming Interface
- AJAX : Asynchronous JavaScript and XML
- CAKB : Cross Awareness Knowledge Base
- CoP : Community of Practice
- CSS : Cascaded Style Sheets
- DBMS : DataBase Management System
- GRDDL: Gleaning Resource Descriptions from Dialects of Languages

- GUI : Graphical User Interface
- HTML: HyperText Markup Language
- HTTP : HyperText Transfer Protocol
- IEEE : Institute of Electrical and Electronics Engineers
- JPEG : Joint Photographic Experts Group
- MVC : Model View Controller
- OO : Object-Oriented
- PNG : Portable Network Graphics
- PSB : Palette Service Browser
- PSP : Palette Service Portal
- PSRF : Palette Service Registry Framework
- RDFS : RDF Schema
- REST : Representational State Transfer
- RIA : Rich Internet Application
- RSS : Really Simple Syndication
- SOAP : Simple Object Access Protocol
- URI : Universal Resource Identifier
- URL : Universal Resource Locator
- UI : User Interface
- XHTML : Extensible HyperText Markup Language
- XML : Extensible Markup Language
- XSLT : (EXtensible Stylesheet Language) XSL Transformations
- W3C : World Wide Web Consortium

1.2 Reading conventions

References by author, like (Fielding, 2000), appear in a bibliography at the end of the deliverable. References by code, like (URL: ALU), appear in a “webography” (list of Web links), also at the end of the deliverable.

2 PALETTE Services search and deployement

PALETTE Services search and deployment answers to the following CoP's concern: "which are the available services? How to find them and use them?", and the answer provided is a common registry for services descriptions coupled with an intuitive interface for browsing it.

PALETTE Services search and deployment is provided in the form of two components, which are: the PALETTE Service Browser (PSB), dedicated to PALETTE services consultation, and the PALETTE Service Registry Framework (PSRF), the main repository where all the available PALETTE Services are registered. The PSB is a web application, which is basically a search portal for PALETTE services. It is the User Interface (UI) to the PSRF, targeted at the community of practice members, and giving them means to discover what are the available services and to deploy these services into their working environment. As the PSRF has already been described in both deliverables D.IMP.04 and D.IMP.05, we will only present the API it provides, which is used by the PSB.

This section thus first details the technical design of the PSB, presenting the information and software architectures, and the PSRF Web Services API. Finally, we present the current version of the PSB, which is available at (URL: PSB).

2.1 Design of the PALETTE Service Browser

The PSB has been designed with the requirement of giving access to the following types of services:

- **Standalone applications**, which are services that must be installed by the end user on its local computer. They are installed by the means of an executable installer file which can be downloaded through the PSB;
- **Web applications**, which are applications that are executed within a browser and which do not require a specific installation. They can be launched directly from the PSB. In some cases these applications may require the creation of a user account;
- **Widgets**, which are small web applications designed to be run in the PALETTE Services Portal container application as described in section 3.2. Some widgets may be linked with a PALETTE Web application, from which they export data or functionalities, and in which case they may require the user to have an account within the Web application. Some others may be independent widgets functioning with, or without, external Web applications or Web services.

Due to the nature of the services addressed, the source of inspiration for the design of the PSB comes from three types of Web applications, which have been studied during the design phase. These applications are:

- Applications (often called widgets, or gadgets) directories found in social network platforms where the user profile page serves as a container that can embed different applications; for instance the Facebook (URL: FCB) application directory;
- Software directories which are available to find desktop applications that can be installed on a personal computer; examples include the download section of the site zdnet.com (URL: ZDN) or the site macupdate.com (URL: MAU);
- Widget directories found on personalized Web Home page sites like netvibes (URL: NVB) and which are similar to the portal described in this report.

The design of the PSB has followed a classical Web design method with the specification of the user interface based on information architecture diagrams and on wireframes of the main pages. The information architecture diagrams represent the different elements of the user interface and the connectors between them. The elements will be translated to pages and the connectors to navigation relationships. They represent the possible navigation flows of the users. The wireframes present the layout and placement of the basic design elements in the user interface. The development team has used the information architecture diagrams and the wireframes to realize the prototype, which is described here. That process is presented in the next sections, as it can be used as an example to guide the design of other types of user interfaces in the PALETTE project.

2.1.1 Information architecture

The information architecture diagrams of the PSB are based on Garrett IA diagrams (Garett, 2002). This is a visual vocabulary for describing interaction which is well documented and for which some templates are even available for certain drawing packages. The application is divided into two main blocks, which are described below.

Service catalog architecture

The service catalog is the entry point of the PSB from where the user gets a list of a compact description of all the available services.

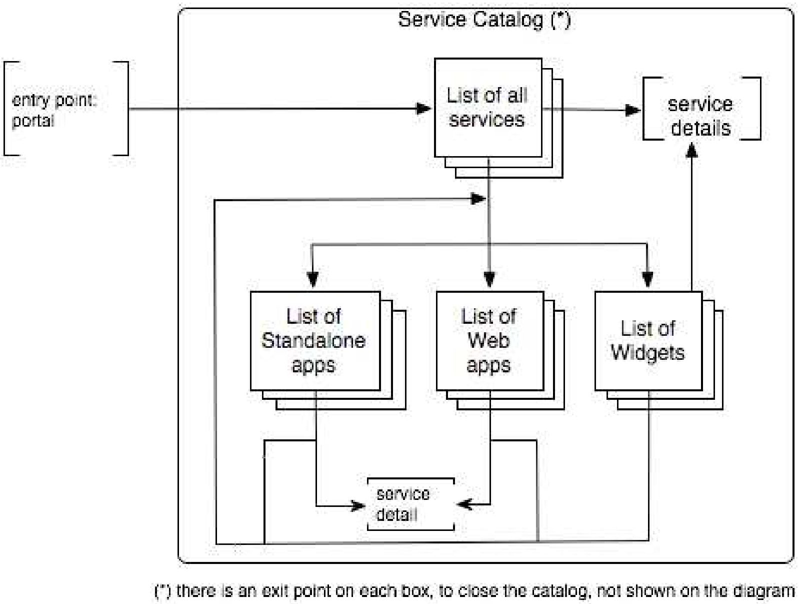


Figure 1: Service catalog architecture

The service catalog is based on four different views, which are very similar:

- Each view is a (eventually multi-pages) list of a short description of a service
- Each view is sorted alphabetically by service name
- Selecting a service from its short description leads to a service information detail view (see below)
- It’s possible to replace one view by any of the 3 others at anytime

The four views are the following ones:

- All services: shows all the services independently of their nature
- Standalone apps: shows only the standalone applications
- Web apps: shows only the Web applications
- Widgets: shows only the widgets

Service detail architecture

The service detail is accessed from the service catalog each time the user selects one type of service to get a more detailed description of it and eventually to install it on her computer.

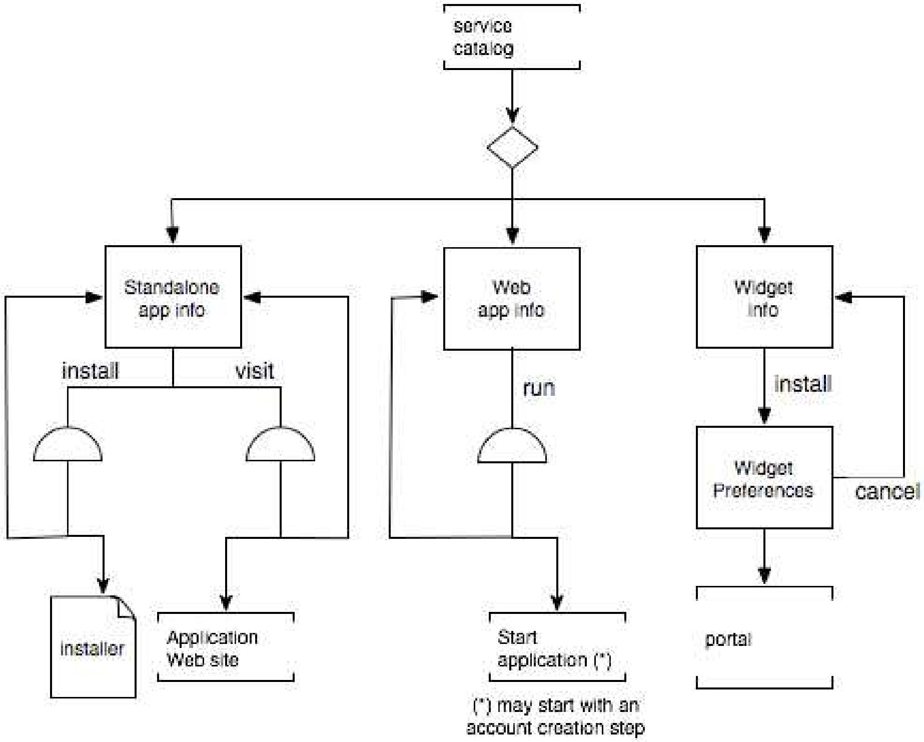


Figure 2: Service detail architecture

The information detail view depends of the nature of the service. However all the views have a similar structure: some service information details are displayed together with some installation instructions and a link to perform the installation.

The action to perform the installation depends of the nature of the application:

- Standalone application: downloads the installer;
- Web application: opens a new window that launches the application;
- Widget: adds the widget to the portal.

The interaction flows may differ depending on the nature of the application:

- Standalone application: it is also possible to visit the application Web site, when it exists, in a new window;

2.1.2 Wireframes

This section presents the generic wireframe that has been defined for each of the service catalog views, and the wireframe defined for the service detail view. The displayed information is not fixed. It is possible to add more information, as soon as it is provided by the PSR.

Service catalog view(s)

The suggested system for navigating between the 4 views of the service catalog is based on a tab group with tabs for each view as illustrated on the next figure.

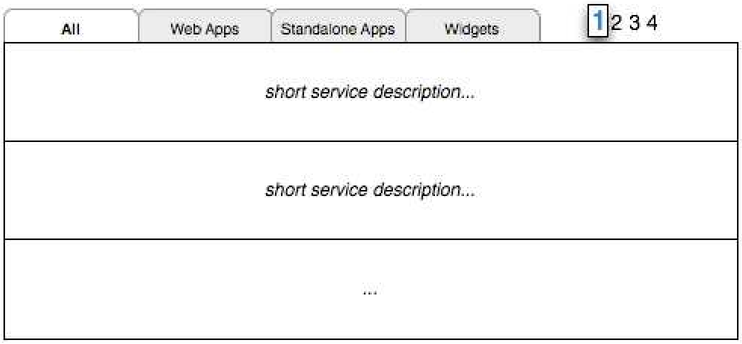


Figure 3: Navigation between views

The suggested window height is of 800 pixels. Navigation is break down into pages if there are more services to display than can fit into the window. The page number is displayed on the view, with the current page highlighted. The user can jump to any page by clicking on its number.

The PSB service short description of services contains the following details, which can be presented like illustrated in Figure 4:

- Service logo (clickable to display the detailed service description view)
- Type of service indication (if catalog view shows all the types of services)
- Service name
- Service provider
- Short textual description (up to *N* characters truncated with '...' if too long, eventually extra characters are displayed in a tooltip window)
- Date of addition into the PSRF
- A details link to display the detailed service description view

Web Application

Cope_It!

By CTI Learning Sector

CoPe_it! is a web-based tool which aims at assisting and enhancing collaboration activities held among members of CoP...

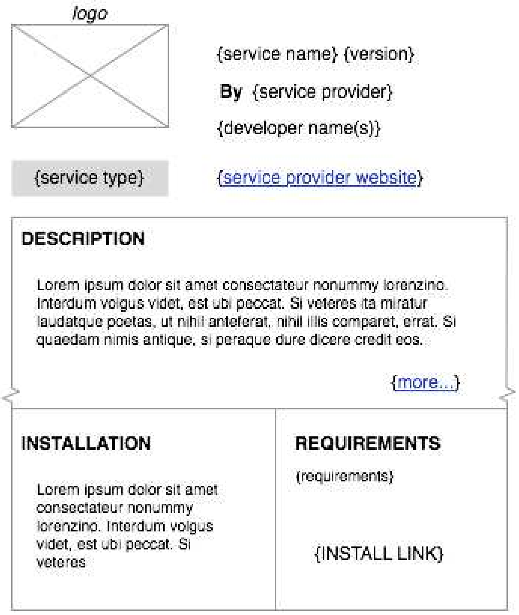
Added *Today*
details

Figure 4: example of presentation of a service short description

Service details view

According to the information contained in the PALETTE services description model, at the time this document has been written, the service detail view can be divided into 4 parts:

- A header with logo, service name, optional version number, service provider, service type (e.g. "Web Application"), optional developer's name, optional service provider website link
- A long description field with an optional "more" link if the service is a standalone application that links to the application web site in a new window
- Installation instructions
- A requirements field that list the requirements necessary for the installation (platforms and web browsers), as well as an install link.



Lorem ipsum dolor sit amet consectateur nonummy lorenzino. Interdum volgus videt, est ubi peccat. Si veteres ita miratur laudatque poetas, ut nihil anteferat, nihil illis comparet, errat. Si quaedam nimis antique, si peraque dure dicere credit eos.

Lorem ipsum dolor sit amet consectateur nonummy lorenzino. Interdum volgus videt, est ubi peccat. Si veteres

Figure 5: Service information details view

The installation instructions are predefined depending of the kind of application considered:

- Standalone app: "Download the application installer with the 'download' link on the side and execute it"
- Web application: "Copy the application URL which is given on the side link into your bookmarks"
- Widget: "Click on the 'Add' button on the side to install the widget into your portal page"

Consequently the {INSTALL LINK} is either:

- A 'Download now' link
- An 'Application url' link
- An 'Add widget' link or button

2.1.3 Software architecture

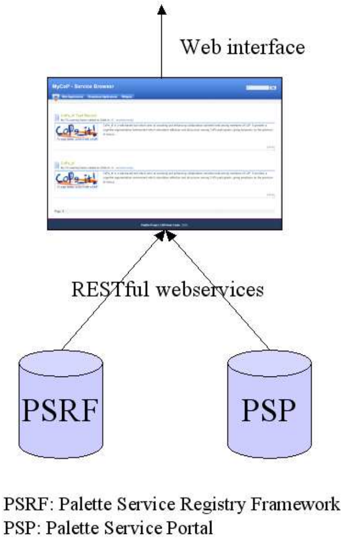

Web interface
RESTful webservices
PSRF
PSP
PSRF: Palette Service Registry Framework
PSP: Palette Service Portal

Figure 6: Data integration architecture

Figure 6 illustrates how data are integrated in the PALETTE Service Browser. Basically the PSB provides access to services registered in the PSRF (URL: PSRF), as well as widgets in the PSP (URL: PSP). Communication between the three applications is done via RESTful web services.

2.1.4 PSRF Web Services API

The PSB access the PSRF through its services API. This one uses REST as communication technology in order to be easily accessible through the Web: client applications are able to invoke the PSRF methods via HTTP POST messages. In particular, PSRF supports two kinds of request messaging based on the format of the exchanged data: The first one (called XML messaging) uses XML formatted Requests and Responses. The second is based on providing simple name-value pairs into the message body, as a simple RESTful service.

The PSRF API comprises a predefined set of web methods that can be used regardless the kind of request messaging. These methods, for which a list is given in annex 8.1, concern the management of both PSRF accounts and PALETTE Service Registrations. In summary, the invocation of PSRF methods allows:

- the PSRF administrators to manage the PSRF accounts,
- the PALETTE Service providers to register and publish their services and,
- the PSRF visitors to search and retrieve information about the available PALETTE Services.

While the PSB is concerned by the last point only, the two others being not dedicated to CoPs can be used directly through HTTP instructions. If necessary, a UI might be developed in the future to facilitate the use of the related functions.

2.2 The PALETTE Service Browser

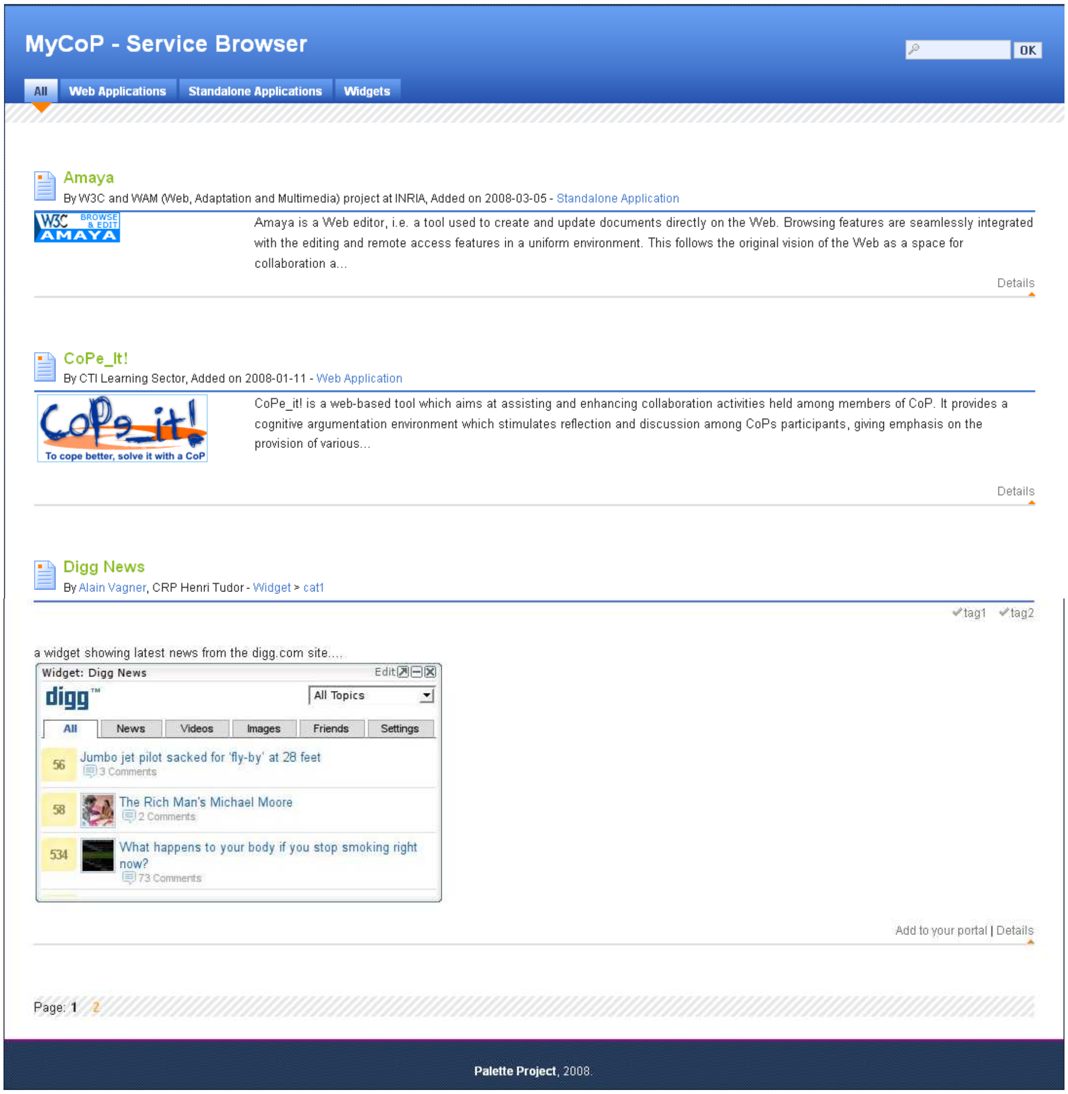
We present here the first version of the PSB, available at (URL: PSB) and its different functions that are made available to users for browsing services in the PSRF and widgets in PSP. The navigation flow and layout follow the specifications of section 2.1.1. Navigation can be performed using tabs for fast access to different filtered views on the available services and widgets list. Some improvements have also been done because there was enough time. First, a trail has been added in order to improve the navigation experience. Second, search functionalities have been introduced based on keywords, tags or categories. The two latter are for the moment provided only for widgets, as they exploit tagging and categorisation functions of the PSP, which are not implemented in the PSRF. Those functions are available in the PSB, only for widgets.

2.2.1 Browsing services

The service browser allows listing all existing services that have been registered in the PSRF. Different ways are proposed to access services details:

- Users can access a service via the “All” tab, which shows the list of all services without any classification way,
- Or access a service by its type (so via the “Web Applications” tab if the service is a web application, or via “Standalone Applications“ or Widgets” if the service is a standalone application or a widget),
- Or using a search function, proposed for each kind of service, which will list services corresponding to given keywords.

All services, web applications, standalone applications views



Amaya

By W3C and WAM (Web, Adaptation and Multimedia) project at INRIA, Added on 2008-03-05 - Standalone Application

Amaya is a Web editor, i.e. a tool used to create and update documents directly on the Web. Browsing features are seamlessly integrated with the editing and remote access features in a uniform environment. This follows the original vision of the Web as a space for collaboration a...

Details

CoPe_It!

By CTI Learning Sector, Added on 2008-01-11 - Web Application

CoPe_it! is a web-based tool which aims at assisting and enhancing collaboration activities held among members of CoP. It provides a cognitive argumentation environment which stimulates reflection and discussion among CoPs participants, giving emphasis on the provision of various...

Details

Digg News

By Alain Vagner, CRP Henri Tudor - Widget > cat1

✓tag1 ✓tag2

a widget showing latest news from the digg.com site....

56 Jumbo jet pilot sacked for 'fly-by' at 28 feet
3 Comments

58 The Rich Man's Michael Moore
2 Comments

534 What happens to your body if you stop smoking right now?
73 Comments

Add to your portal | Details

Page: 1 2

Palette Project, 2008.

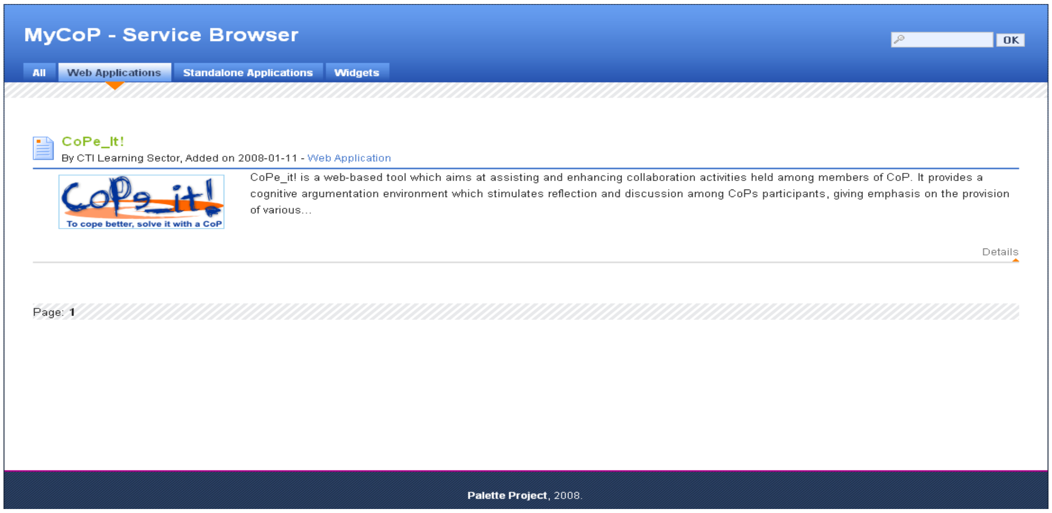
Figure 7: Browsing all services

When first accessing the service browser, the first view one can see is the “all services” view (Figure 7). All services are listed whatever their kind. For each service listed, the displayed information is:

- The service name
- Its owner
- The date it has been added to the list
- Its type
- Its logo
- A description of the service

Additionally for each one, a “details” link is provided.

Figure 8 and Figure 9 show the views for respectively web and stand-alone applications, which are filtered versions of the all services view.



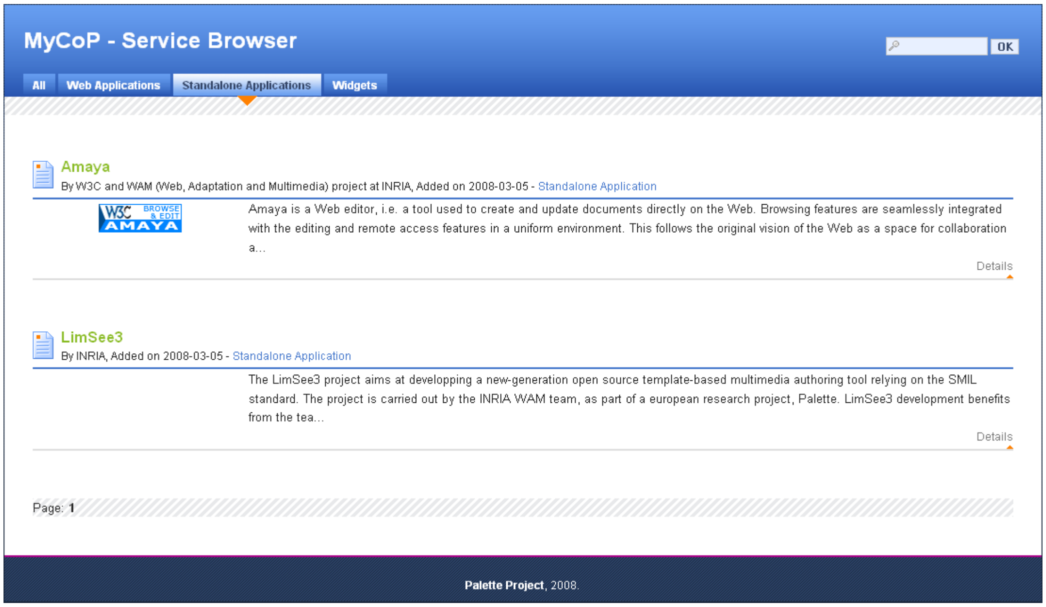
CoPe_It!

By CTI Learning Sector, Added on 2008-01-11 - Web Application

CoPe_it! is a web-based tool which aims at assisting and enhancing collaboration activities held among members of CoP. It provides a cognitive argumentation environment which stimulates reflection and discussion among CoPs participants, giving emphasis on the provision of various...

Palette Project, 2008.

Figure 8: Browsing Web applications



Amaya

By W3C and WAM (Web, Adaptation and Multimedia) project at INRIA, Added on 2008-03-05 - Standalone Application

Amaya is a Web editor, i.e. a tool used to create and update documents directly on the Web. Browsing features are seamlessly integrated with the editing and remote access features in a uniform environment. This follows the original vision of the Web as a space for collaboration a...

LimSee3

By INRIA, Added on 2008-03-05 - Standalone Application

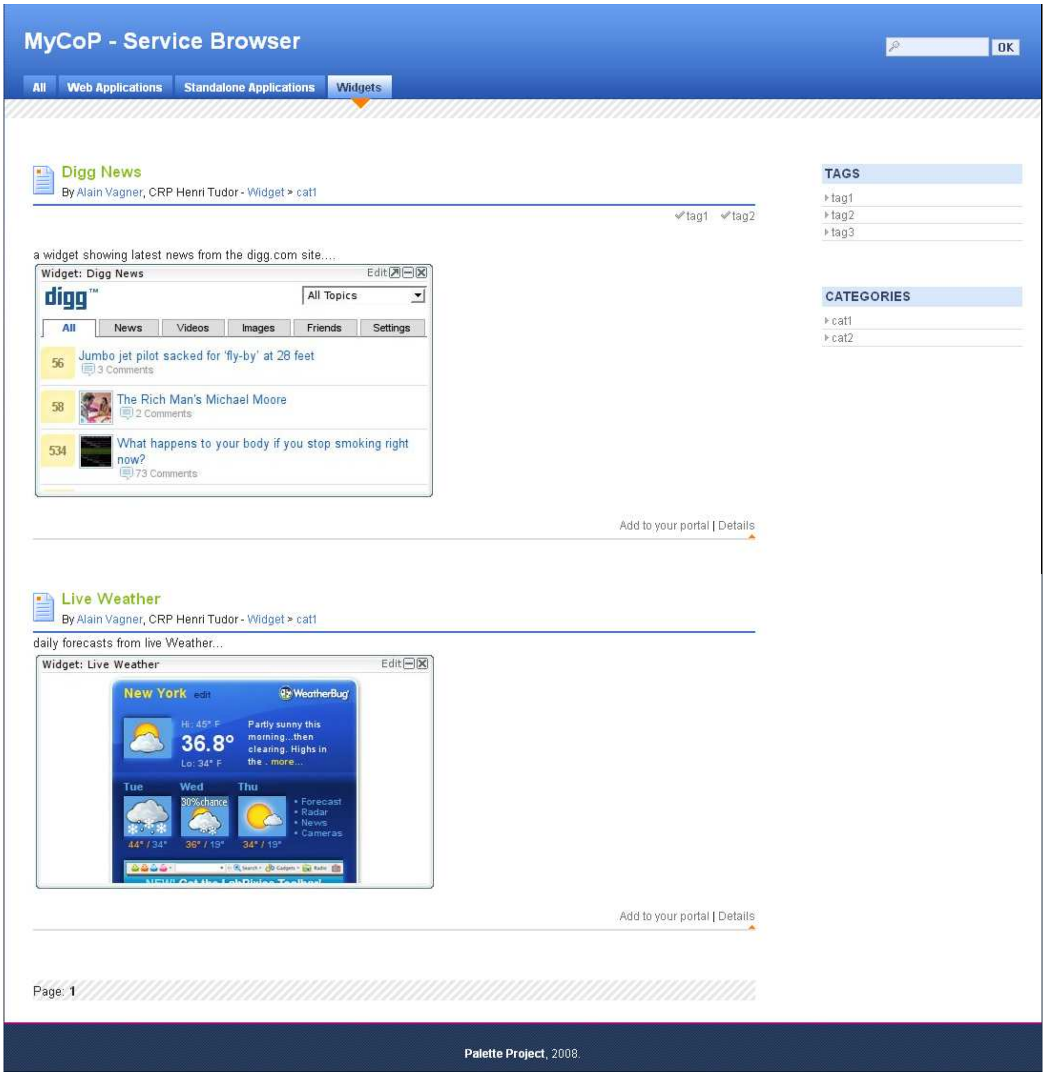
The LimSee3 project aims at developping a new-generation open source template-based multimedia authoring tool relying on the SMIL standard. The project is carried out by the INRIA WAM team, as part of a european research project, Palette. LimSee3 development benefits from the tea...

Palette Project, 2008.

Figure 9: Browsing Standalone applications

Widgets view

As shown in Figure 10, the view provided for widgets shows some specific details.



MyCoP - Service Browser

Digg News

By Alain Vagner, CRP Henri Tudor - Widget > cat1

a widget showing latest news from the digg.com site....

Widget: Digg News

56 Jumbo jet pilot sacked for 'fly-by' at 28 feet — 3 Comments

58 The Rich Man's Michael Moore — 2 Comments

534 What happens to your body if you stop smoking right now? — 73 Comments

Live Weather

By Alain Vagner, CRP Henri Tudor - Widget > cat1

daily forecasts from live Weather...

Widget: Live Weather

New York — Hi: 45° F — 36.8° — Lo: 34° F — Partly sunny this morning...then clearing. Highs in the . more...

Tue 44° / 34° — Wed 30% chance 36° / 19° — Thu 34° / 19°

Palette Project, 2008.

Figure 10: Browsing widgets

For each widget, the displayed information is:

- It name
- Its Author
- Its description
- A preview

Actions that can be performed by the user are the following:

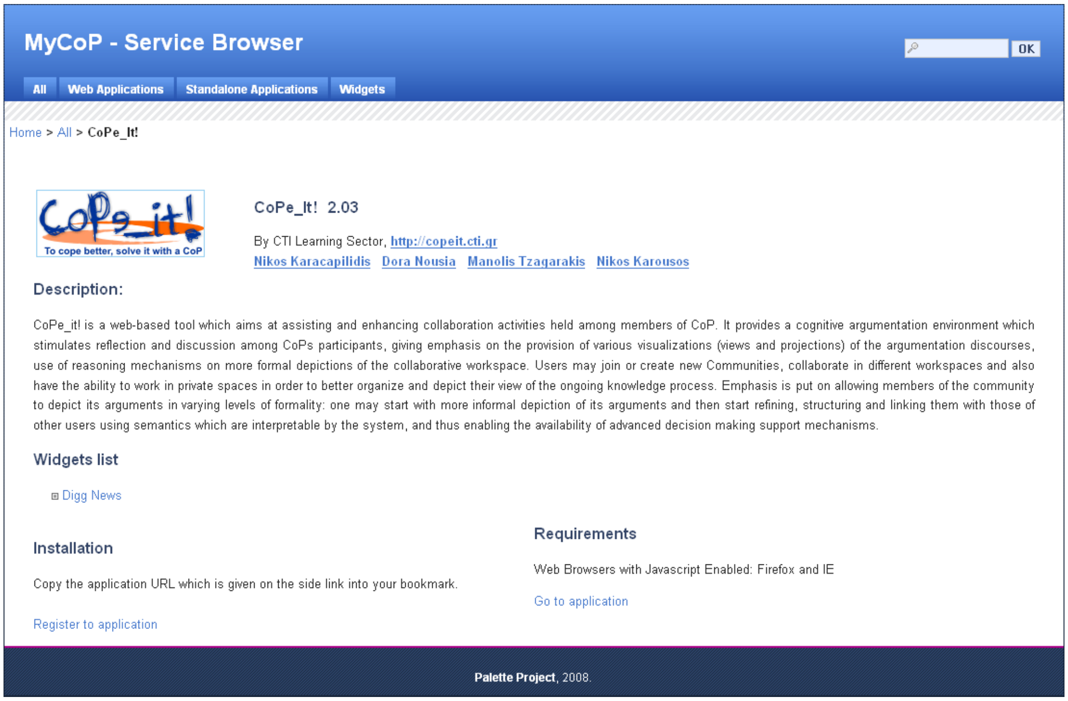
- Add it to the portal (PSP),
- Access the widget's details

Beside the widgets list can be found the list of tags and categories of all widgets. This allows searching widgets by categories or tags. More information is available in section 2.2.3.

2.2.2 Viewing service details

Web and standalone applications details view

Details concerning each service can be obtained by clicking on the “Details” link.



CoPe_It! 2.03

By CTI Learning Sector, http://copeit.cti.gr
Nikos Karacapilidis Dora Nousia Manolis Tzagarakis Nikos Karousos

Description:

CoPe_it! is a web-based tool which aims at assisting and enhancing collaboration activities held among members of CoP. It provides a cognitive argumentation environment which stimulates reflection and discussion among CoPs participants, giving emphasis on the provision of various visualizations (views and projections) of the argumentation discourses, use of reasoning mechanisms on more formal depictions of the collaborative workspace. Users may join or create new Communities, collaborate in different workspaces and also have the ability to work in private spaces in order to better organize and depict their view of the ongoing knowledge process. Emphasis is put on allowing members of the community to depict its arguments in varying levels of formality: one may start with more informal depiction of its arguments and then start refining, structuring and linking them with those of other users using semantics which are interpretable by the system, and thus enabling the availability of advanced decision making support mechanisms.

Widgets list

- Digg News

Installation

Copy the application URL which is given on the side link into your bookmark.

Register to application

Requirements

Web Browsers with Javascript Enabled: Firefox and IE

Go to application

Palette Project, 2008.

Figure 11: Browsing a service details

For each service, the following details are displayed like illustrated by Figure 11:

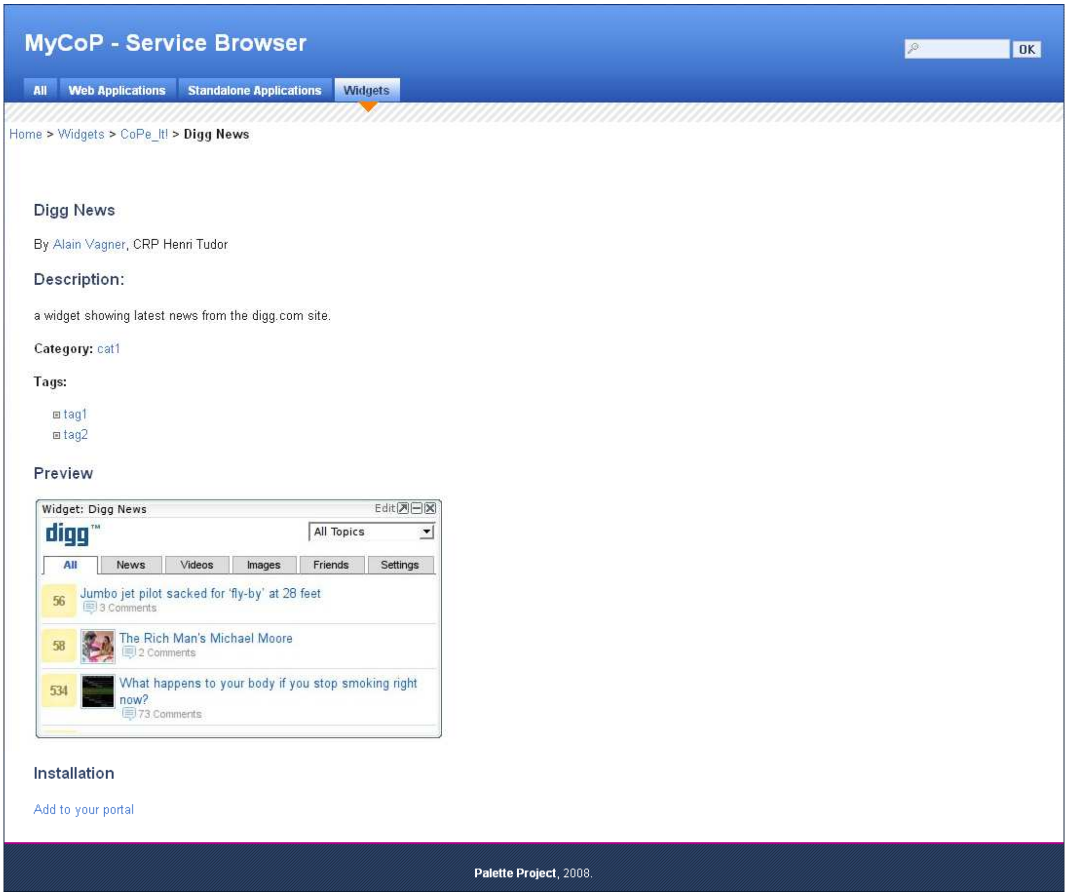
- In the header of the description: its logo, title, author and developers
- Then its description. A “more” link enables to access for example the service website
- The widgets list allowing seeing the list of widgets associated to the service. Clicking on it will redirect to the widget description.
- The installation part gives instructions on the way to install the service, and proposes a link to register to the application.
- The requirements part gives information about requirements such as general information, or supported operating systems. It is there you will find the “download now” link.

Widget details view

The user can access a widget details by different ways:

- By the “All” services list, selecting a widget
- By the “Widgets” list
- By the keyword search way, searching for a particular keyword
- By the tags or categories search way (detailed later)
- Selecting a service linked to other widgets

In each case, the widget details will be shown the same way, as illustrated in Figure 12.



Home > Widgets > CoPe_It! > Digg News

Digg News

By Alain Vagner, CRP Henri Tudor

Description:

a widget showing latest news from the digg.com site.

Category: cat1

Tags:

- tag1
- tag2

Preview

56 Jumbo jet pilot sacked for 'fly-by' at 28 feet
3 Comments

58 The Rich Man's Michael Moore
2 Comments

534 What happens to your body if you stop smoking right now?
73 Comments

Installation

Add to your portal

Palette Project, 2008.

Figure 12: Viewing a widget details

In a widget detail view, the user finds:

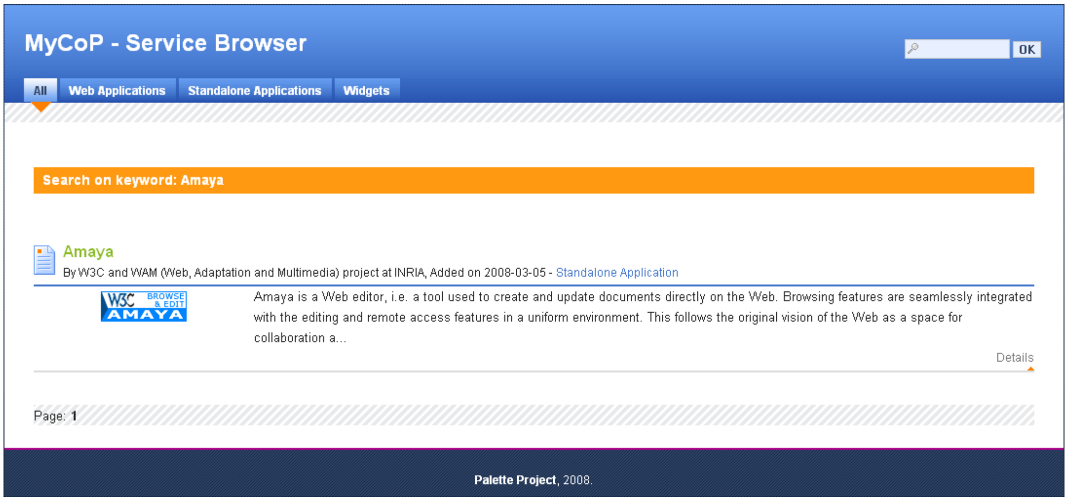
- The widget name
- Its author
- Its description
- Its category and tags (used for the search on widgets by categories and tags)
- A preview

As in the non-detailed view, an installation part proposes a link that enables adding the widget to a PSP.

2.2.3 Searching services

Keyword searching

A keyword search is available at the top of the service browser. It has been added to the specification, as it seemed a natural improvement. This search combines the PSRF search and the PSP (URL: PSP) search web services. More advanced search functions (e.g. by category or service property) might be developed in the future. Figure 13 provides an example for a search with the keyword “Amaya”.



MyCoP - Service Browser

Search on keyword: Amaya

Amaya

By W3C and WAM (Web, Adaptation and Multimedia) project at INRIA, Added on 2008-03-05 - Standalone Application

Amaya is a Web editor, i.e. a tool used to create and update documents directly on the Web. Browsing features are seamlessly integrated with the editing and remote access features in a uniform environment. This follows the original vision of the Web as a space for collaboration a...

Details

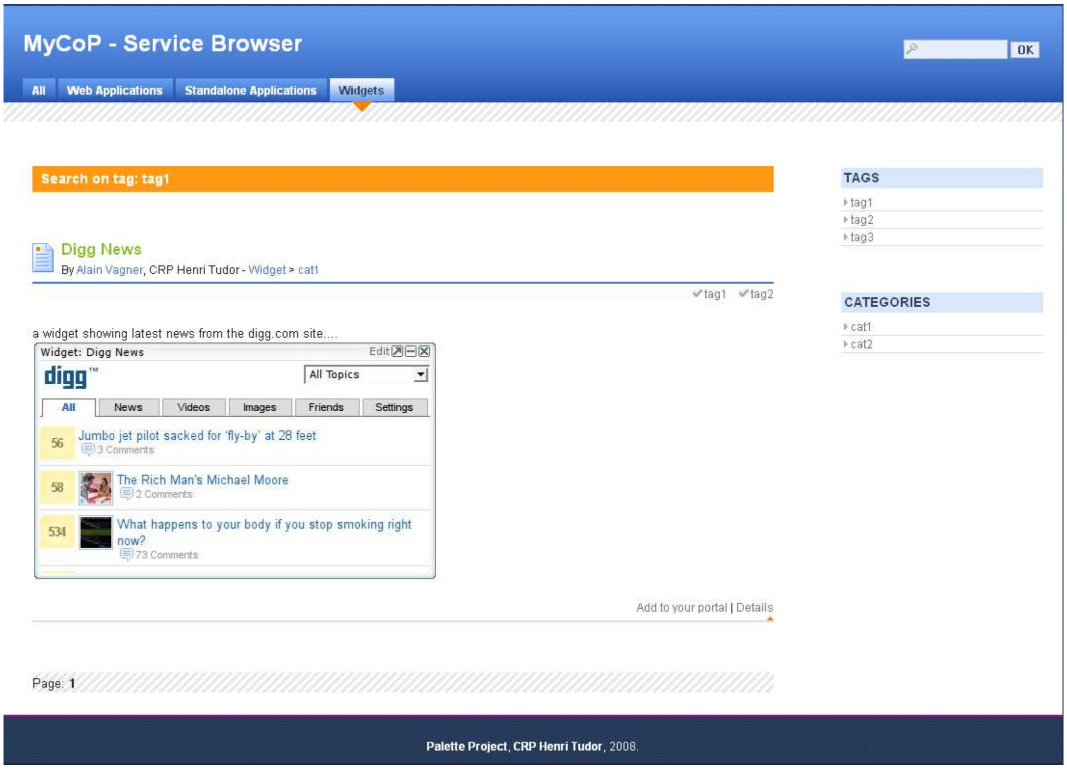
Page: 1

Palette Project, 2008.

Figure 13: Search on keyword "Amaya"

Searching widgets by tag

The search by tag is only available for widgets, providing the list of widgets that have been tagged with one given tag. Figure 14 shows the results of a search on a “tag1” tag.



MyCoP - Service Browser

Search on tag: tag1

Digg News

By Alain Vagner, CRP Henri Tudor - Widget > cat1

✓tag1 ✓tag2

a widget showing latest news from the digg.com site....

Widget: Digg News

56 Jumbo jet pilot sacked for 'fly-by' at 28 feet
3 Comments

58 The Rich Man's Michael Moore
2 Comments

534 What happens to your body if you stop smoking right now?
73 Comments

Add to your portal | Details

Page: 1

TAGS
- tag1
- tag2
- tag3

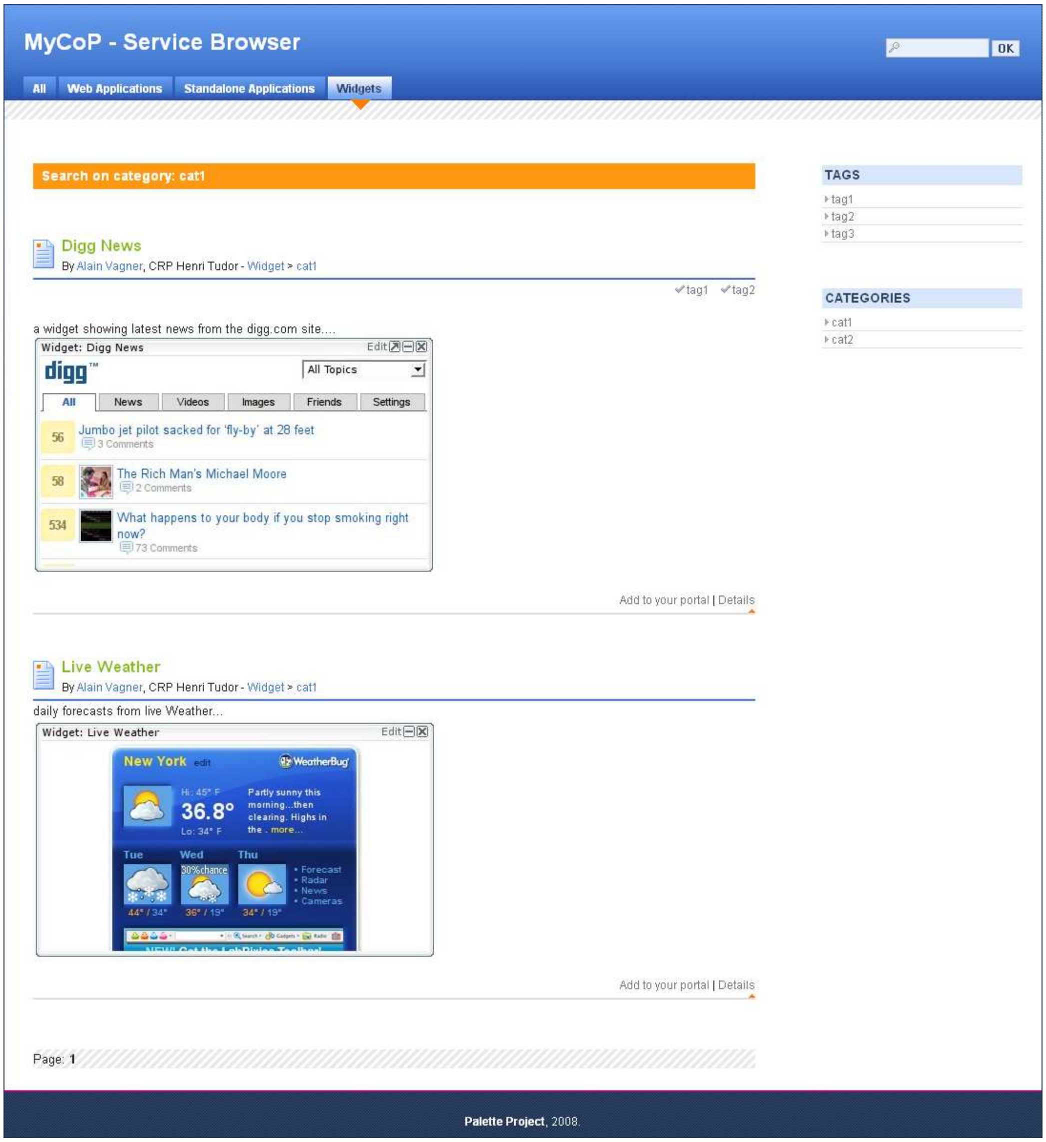
CATEGORIES
- cat1
- cat2

Palette Project, CRP Henri Tudor, 2008.

Figure 14: Search widgets by tag

Searching widgets by category

The search by category is only available for widgets, providing the list of widgets that have been categorized in one given category. Figure 15 shows the results of a search on the “cat1” category.



Search on category: cat1

Digg News

By Alain Vagner, CRP Henri Tudor - Widget > cat1

tag1 tag2

a widget showing latest news from the digg.com site....

Widget: Digg News

Jumbo jet pilot sacked for 'fly-by' at 28 feet
3 Comments

The Rich Man's Michael Moore
2 Comments

What happens to your body if you stop smoking right now?
73 Comments

Live Weather

By Alain Vagner, CRP Henri Tudor - Widget > cat1

daily forecasts from live Weather...

Widget: Live Weather

Partly sunny this morning...then clearing. Highs in the ... more...

Palette Project, 2008.

Figure 15: Search widgets by category

3 PALETTE services integration in the presentation layer

The presentation layer is the last layer in the n-tiered architecture of Web services. It makes the interface between users and the services. In particular, that's in the presentation layer that resides the User Interface of a service. The presentation layer assures the integration of end users inputs, its transformation into data and commands sent to the service, and the visualization of the service outputs.

In a Service-Oriented Architecture the software unit is a component, which is combined by loose coupling connections with other software components. In this model, the integration at the presentation layer is the integration of components by combining their presentation front ends instead of their application logic or data schemas (Daniel & al., 2007).

In the services considered so far in PALETTE, the entire user interfaces are graphical user interfaces (GUIs). Apart from the provision of multimedia documents, these GUIs do not involve interaction techniques such as multimodal interaction. Thus the integration in the presentation layer is in fact the visual integration of the user interfaces of the components that constitute the services. Visual integration deals in particular with the dynamical allocation of mouse and keyboard inputs to different components and to the different means to share screen real estate between theses components.

This section is divided into two parts. The first part presents a state of the art of visual integration methods, together with some suggestions of application to PALETTE services. The second part elaborates a visual and technical solution for integration in the presentation layer: the portal, which we have decided to implement in PALETTE.

This section is focused on the services, which are available as Web applications. For standalone applications, we do not consider other integration in the presentation layer than the means provided by each operating system and which are known as the "desktop metaphor" from which users can launch several applications in several windows at the same time.

3.1 On WEB visual integration methods

The GUI of a Web application is displayed inside a Web browser window. Thus, the majority of the methods for visual integration allocate the window space to the different components by sharing it either in adjacent spaces, in layers or in time. Some methods are permanent in the sense that the visual integration is set-up as soon as the services are started, while some other methods are dynamical in that an initial service can be extended dynamically. The following sections described 9 different methods.

3.1.1 The tab group

The tab group is a basic form of visual composition of services. It works by nesting several views inside a container view. It is available in any modern browser that supports tab browsing in which different pages can be opened in the same window. The current page is selected by clicking on its corresponding tab at the top of the window. This form of visual integration is adapted to improve the navigation efficiency when passing from one application to another one.

The tab group can also be simulated directly on behalf of the services. That design option is more common as a primary navigation menu within a single service, but it is possible to apply it to display different services hosted on different servers in the different tabs. This is the case with the Plaxo social network services (URL: PSN). As you can see on the illustrations below, the "Groups" and the "Address Book" services are hosted on different servers.

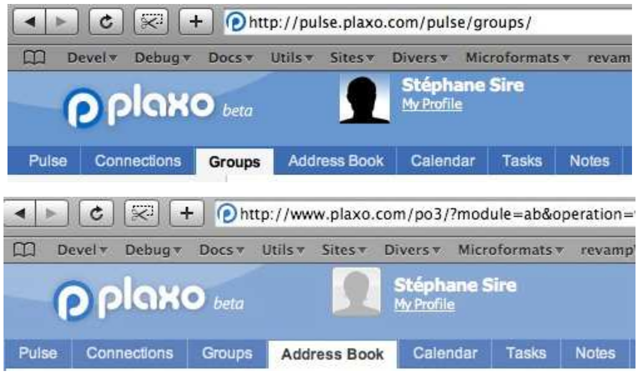
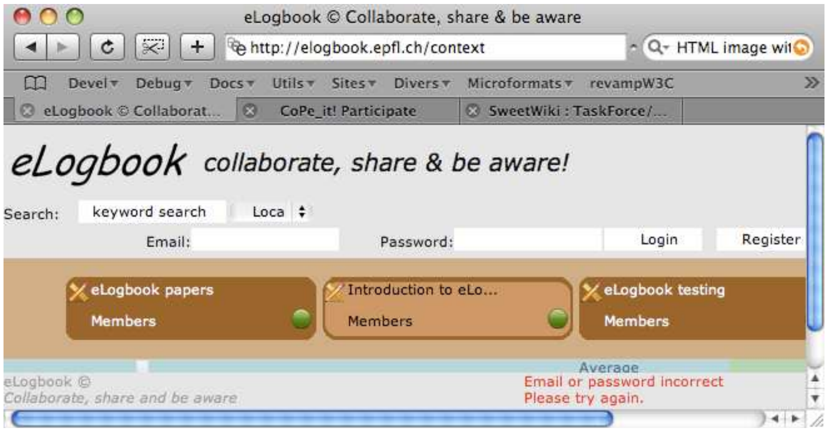


Figure 16: Two services "Groups" and "Address Book" integrated with a tab group

Tab groups are implemented with a conjunction of XHTML elements and CSS layout instructions, each service displaying the same tab headers, or by other means such as HTML frames. In that later case, the nested views can be full applications windows, which are reframed to fit inside their container's view.

Applicability to PALETTE

The figure below shows what a tab group view of PALETTE services would look like. In fact it is already available in any Web browser that support tab browsing, without modifying any service.

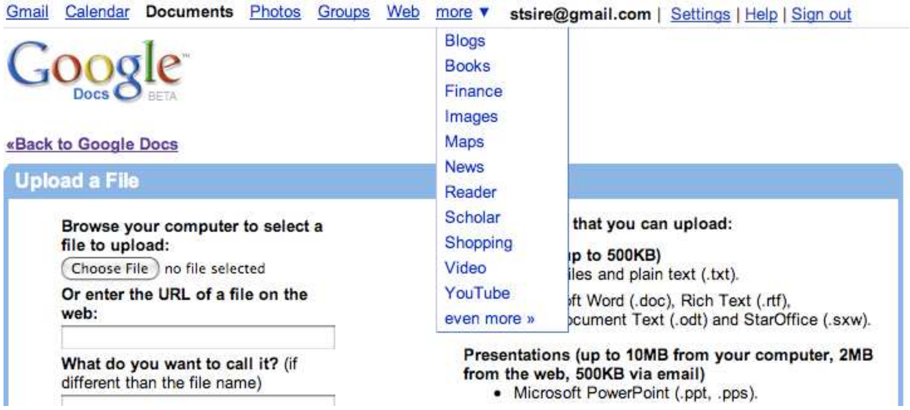


Email or password incorrect
Please try again.

Figure 17: e-LogBook, Cope_It! and SweetWiki in the same window using browser tab navigation

3.1.2 The menu bar

The menu bar is an alternative to a tab group which aims at improving navigation efficiency too. This is illustrated by Google suite of Web applications, which is visually integrated through the presence of a menu bar displayed at the top of each application (URL: GSA). Like the tab group the main purpose of the menu bar design is to improve the navigation experience.



Browse your computer to select a file to upload:

no file selected

Or enter the URL of a file on the web:

What do you want to call it? (if different than the file name)

that you can upload:

ıp to 500KB)

iles and plain text (.txt).

ıft Word (.doc), Rich Text (.rtf),

ocument Text (.odt) and StarOffice (.sxw).

Presentations (up to 10MB from your computer, 2MB from the web, 500KB via email)

- Microsoft PowerPoint (.ppt, .pps).

Figure 18: Google menu bar at the top to navigate from one Web application to another

Applicability to PALETTE

The menu bar design could be developed for PALETTE. That would require either to have each service to use the same instructions to display the menu bar at the top, or to develop a new application just for displaying the menu bar and that would launch the services inside internal frames. The first solution is costly to implement because it requires to develop a specific version of each service to be run in the integrated environment. The second solution may not be quite usable if the individual services also have their own navigation bar at the top. This design fits well in a context where a single sign on mechanism allows the user to navigate from one service to another without re-entering authentication information. In that context, it gives the illusion of a single application.

3.1.3 The service embedded as an image

A service can easily embed an image generated by another service, such as JPEG or PNG image, in its content. This composition mechanism is quite simple to implement as it only requires to define an API for the service that generates the image, and to embed the service call as an image tag. For instance, the Google Chart API (URL: GCA) defines an API for services that can generate charts which is followed by many pie Web services.

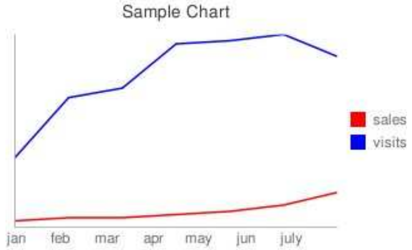


Figure 19: A chart generated as an embedded image from Google Chart service

The external service invocation is performed through the href attribute of the image tag. That method is limited to calling services through a HTTP GET method. The parameters are passed with the image URL. The image above has been generated with the following image tag:

```
<img
src="http://chart.apis.google.com/chart?chs=320x200&cht=lc&ch
tt=Sample%20Chart&chd=s:CDDEFHL,Wps6792&chco=ff0000,0000ff
&chdl=sales|visits&chxl=0:|jan|feb|mar|apr|may|jun|%20july|&c
hxt=x"/>
```

Applicability to PALETTE

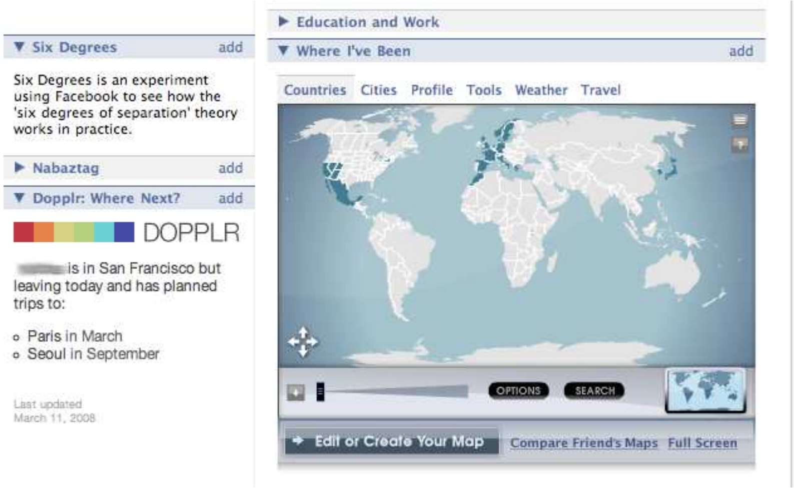
This composition method is limited to services that can export information, which is meaningful in a graphical format. It is also limited to the integration of non-interactive data. Currently the most graphical service of PALETTE, Cope_It!, could use it, for instance to export some graphical representation of a workspace argumentation inside another service, such as within a Wiki page. In the future, as Web browsers support interactive graphical formats such as SVG, this method will become an alternative to the next method.

3.1.4 The service embedded as a widget and the Portal

The concept of a customizable portal with news, stock quotes, weather, and access to many other "miniaturized" services has been popularized during the last years with mainstream Web portals such as Netvibes (URL: NVB). In a customizable portal, users can manually place miniaturized versions of services on a grid with drag and drop operations. The miniaturized version is called a Widget. Users can eventually group the widgets by pages, or workspaces accessible through a tab group. Each widget stores the value of several user-defined properties between sessions. The widget version of the services have to comply with a specific portal API.

In addition to portals, there are other kinds of container applications that can execute widgets. For instance social network home pages, such as FaceBook (URL: FCB), allow the integration of widgets on the user's profile page that is seen by the user and by her friends. In the future, as common API for the visual integration of widgets and for access to end user data are adopted, such as The OpenSocial initiative (URL: OSA), a widget will be able to run on several different platforms.

The illustration below shows the “Where I’ve been” widget, in Facebook and other widgets. The widgets can be collapsed, or visible, or sometimes they can be displayed in full window mode. Some widgets are just front-ends to external services which are hosted outside of Facebook and can also be accessed through a traditional Web application.



Six Degrees is an experiment using Facebook to see how the 'six degrees of separation' theory works in practice.

[redacted] is in San Francisco but leaving today and has planned trips to:

- Paris in March
- Seoul in September

Last updated
March 11, 2008

Figure 20: Extract of a user’s profile page on Facebook that shows several widgets

Currently, the integration of services in portals remains purely visual. The widgets are independent one from the other. To our knowledge portal implementations do not provide widget developers with more advanced API to support single sign on between services, or to share data. The portal remains a purely graphical host (or container). For the same reason there is currently no portal supporting drag and drop operations between widgets.

Customizable portals and social network applications come with a widget browser service similar to the PALETTE Service Browser described in this report.

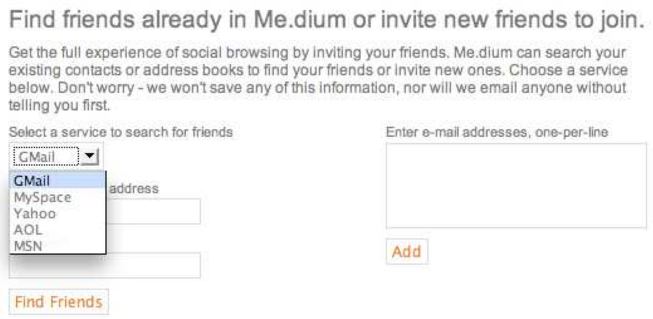
Applicability to PALETTE

As described later in this chapter, the portal is a convenient solution to propose a host environment for the integration of widgets offering access to different PALETTE service functionalities. At the most basic level, we can imagine a portal that displays a composition of the RSS feeds provided by each service. At the most advanced levels, we can imagine a portal that allows to drag and drop files from a service to another one, producing a kind of Web desktop metaphor. A main advantage of the portal is that it can be customized by the end users to fit particular tasks by a selection of adequate widgets.

3.1.5 The data importation widget

A data importation widget represents the integration between services at the data layer in the presentation layer. It is directly displayed into a data input form of a service when the data can be imported from another service. This is frequent for instance in social network sites, where it is quicker to import the list of a user's friends than to type it again in a second site.

Visually, the data import widget can be presented as a drop down list that proposes a list of compatible services from which to import data. The widget displayed in the figure below is placed directly in the registration form. It allows selecting a service from which the user is already member and from which the user will ask the service to retrieve data.



Find friends already in Me.dium or invite new friends to join.

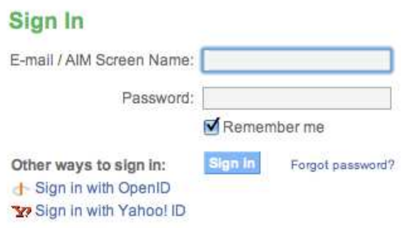
Get the full experience of social browsing by inviting your friends. Me.dium can search your existing contacts or address books to find your friends or invite new ones. Choose a service below. Don't worry - we won't save any of this information, nor will we email anyone without telling you first.

Select a service to search for friends: GMail

Enter e-mail addresses, one-per-line

Figure 21: Data import widget in a social network site (URL: MED)

The single sign on process based on OpenID uses now a well-known icon, which can be considered as a special data import widget. It triggers the importation of the identification data from an external identification server, which allows users to have a single identifier for all the services she uses. In fact it consists of an OpenID icon/link usually placed next to the sign in or register dialog. When clicking on the OpenID icon the window content is replaced (or a new window is opened) that requests the user her identification. At the end of the identification dialog the window content is redirected back to the initial service from where the identification was requested.



Sign In

E-mail / AIM Screen Name:

Password:

☑ Remember me

Figure 22: A sign in dialog with an OpenID action link on the bottom left

Many types of data can be shared and imported. Another popular example concerns the importation of photos in photo manipulation services such as Picnik (URL: PIC) from photo sharing service such as Flickr (URL: FLK). In that case the photo can be saved back into the original service after transformation.

Applicability to PALETTE

We have already described in the deliverable D.IMP.04 the cross-service document importer widget that could be developed in PALETTE to import documents from within services. The single sign on service has also been recognized as useful and will require the introduction of the OpenID action link.

3.1.6 The data exportation widget

Symmetrically to a data import widget, a data exportation widget triggers a service invocation method that exports data or performs an action with that data into a target service.

A very frequent application of that principle is to add a social bookmarking widget inside a page. All the links or the buttons in the widget export a link to the current page into a corresponding social bookmarking site the user has an account into. The snapshot below shows the "AddThis" social bookmarking widget. When clicked, it opens a menu to select the bookmarking site (URL: ATB). However the integration is not seamless for end users as it requires an intermediate step where she has to enter her login/password for the tagging service.

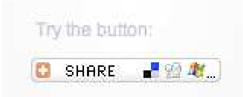
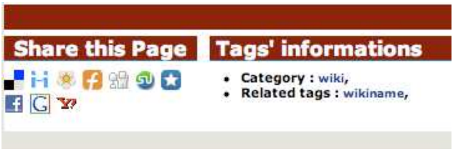


Figure 23: AddThis social bookmarking widget

Many data exportation services also allow to select contextual information by selecting text on the page, and to send it to the target service. For instance a description of the tagged resource can be directly copied from the current text selection on the page.

Applicability to PALETTE

Any service that can produce/consume data which can be consumed/produced by another service is susceptible to incorporate data import/export widgets. For instance a prototype of a social bookmarking widget has been done in SweetWiki in WP3.



- Category : wiki,
- Related tags : wikiname,

Figure 24: Prototype for a social bookmarking widget to be added at the bottom left of each SweetWiki page.

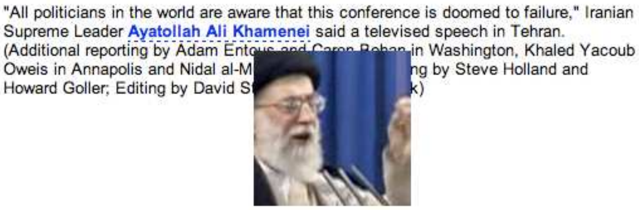
3.1.7 The data mashup and micromashup

The mashup is a visual integration and a data composition style. In a mashup, data from one or more services is aggregated and displayed by one service using one or more views, which can be overlaid or adjacent. The views are synchronized and when data coming from one source changes or when the user requests a change, such as selecting an item in one view, the other views are updated accordingly. As explained in D.IMP.04, a data mashup is usually a Web application served from a mashup host that allows data sources to be combined from different servers.

The micromashup is a special case of mashup where one source of data is the page itself. The data contained in the page is extracted (or scraped) with a transformation style sheet or a script written in XSLT or in Javascript and that is executed on the client side. The extraction can be facilitated if the page contains microformat data. The W3C GRDDL recommendation describes a syntax for attaching an extraction style sheet to a page (Connolly, 2007).

The **overlay** mashup is a layer-by-layer visual composition technique: the data is displayed into different layers using transparency and alignment methods. The best illustration is the Google map widget (URL: GMA). This widget can be inserted in a page and display additional data coming from one or more services on top of the map. The data is overlaid either directly onto the map, or it can be displayed into information bubbles that can be opened or closed from small point of interest icons added on top of the map.

The widget overlay can contain information in any modality. For instance, the facial video recognition service Viewdle (URL: VWD) displays a recent video of people whose name are displayed in a page, on mouse over the name. It uses a database of well-known people, matching their name with their photo, and is able to automatically detect the people faces inside video streams, such as the streams published every day by news agencies like Reuters.

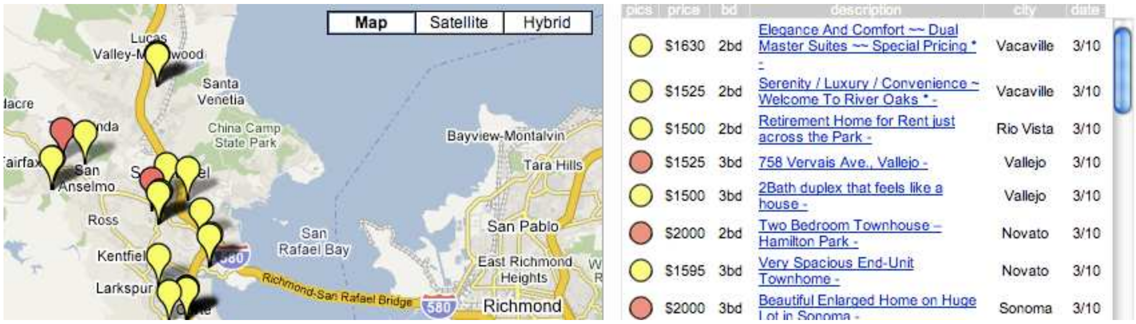


"All politicians in the world are aware that this conference is doomed to failure," Iranian Supreme Leader Ayatollah Ali Khamenei said a televised speech in Tehran. (Additional reporting by Adam Ent in Washington, Khaled Yacoub Oweis in Annapolis and Nidal al-M ng by Steve Holland and Howard Goller; Editing by David S k)

Figure 25: Viewdle adds a layered mashup view of a recent video showing the face of a person whose name is cited in the text

Viewdle proposes a name widget service that anybody can embed into one's blog for instance. Using that service, one can specifically tag all the people names she cites in a blog entry with a microformat class attribute. Then, by loading a viewdle Javascript library into the page, all the people name detected will be turned into special types of links that will trigger the display of a video, cropped to the people face, when the user moves the mouse pointer over the name.

The **juxtaposition** mashup is a side-by-side view of complementary data. For instance the site Housingmaps.com (URL: HMC) displays lists of apartments from Craigslist (URL: CGL) next to a Google map of their location[1].



pics	price	bd	description	city	date
🟡	$1630	2bd	Elegance And Comfort ~~ Dual Master Suites ~~ Special Pricing * -	Vacaville	3/10
🟡	$1525	2bd	Serenity / Luxury / Convenience ~ Welcome To River Oaks * -	Vacaville	3/10
🟡	$1500	2bd	Retirement Home for Rent just across the Park -	Rio Vista	3/10
🔴	$1525	3bd	758 Vervais Ave., Vallejo -	Vallejo	3/10
🟡	$1500	3bd	2Bath duplex that feels like a house -	Vallejo	3/10
🔴	$2000	2bd	Two Bedroom Townhouse – Hamilton Park -	Novato	3/10
🟡	$1595	3bd	Very Spacious End-Unit Townhome -	Novato	3/10
🔴	$2000	3bd	Beautiful Enlarged Home on Huge Lot in Sonoma -	Sonoma	3/10

Figure 26: Mashup with juxtaposition of a map, left, and a list of geolocalized items

There is a difference between mashup juxtaposition and a portal composition: usually the widgets displayed in a portal are not synchronized together: when a portal widget changes of state, the other widgets on the same page do not change. However that will certainly evolve in the future, with portal supporting mashup composition style of widgets. For instance, the W3C widget draft specification is slowly introducing a mechanism for cross-widget message posting (Van Kesteren & al., 2007).

[1] You can notice that Housingmaps.com also displays the position of each apartment with a map overlay.

Applicability to PALETTE

As several PALETTE services will export information about content data or content as RSS feeds, it is conceivable to develop data mashup on top of these data streams. However, up to now none of these feeds will contain geographically tagged data, which does not make map based mashups relevant. Currently only the SweetWiki service contains explicit microformat content, which makes it a good candidate to build micromashups.

3.1.8 The Browser extension

The browser extension is a visual integration and a programming technique. It allows to make modifications to the browser UI itself. Modifications can consist in adding some menus or menu options to the browser menu bar, adding some sidebars which can be displayed on the side of any browser window, adding some action buttons or popup menus in the status bar or in the toolbar of any browser window, etc. It is also possible to directly change on the fly the content of the page. The modifications can be permanent or contextual. In the latter case they may depend on the URL or on the content of the page, which is displayed.

Each browser supports its own extension method. Hundreds or extensions are available for Firefox (URL: FAO). Some extensions completely redesign an existing Web site, for instance to improve its user interface such as the Better Gmail Firefox extension (URL: BGF).

The illustration below shows Operator (URL: OPR), a Firefox extension visible as a menu bar displayed on top of each window. The extension detects microformated data into the window content with pre-registered detectors. For each data type recognized, Operator activates a specific menu and fills it with actions. Pre-defined detectors works for addresses, contacts, events, locations, tags names, etc. The user can install new detectors. From each menu, the user can invoke one or more external services on the selected data. These services may eventually open new windows.

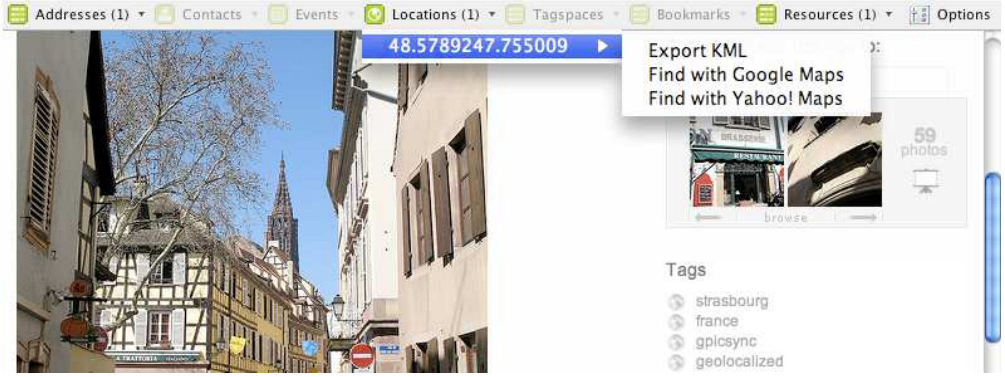


Figure 27: Operator toolbar with a Flickr page that contains a geo tagged photo; Operator has detected the Address and Location of the photo and proposes specific actions in popup menus.

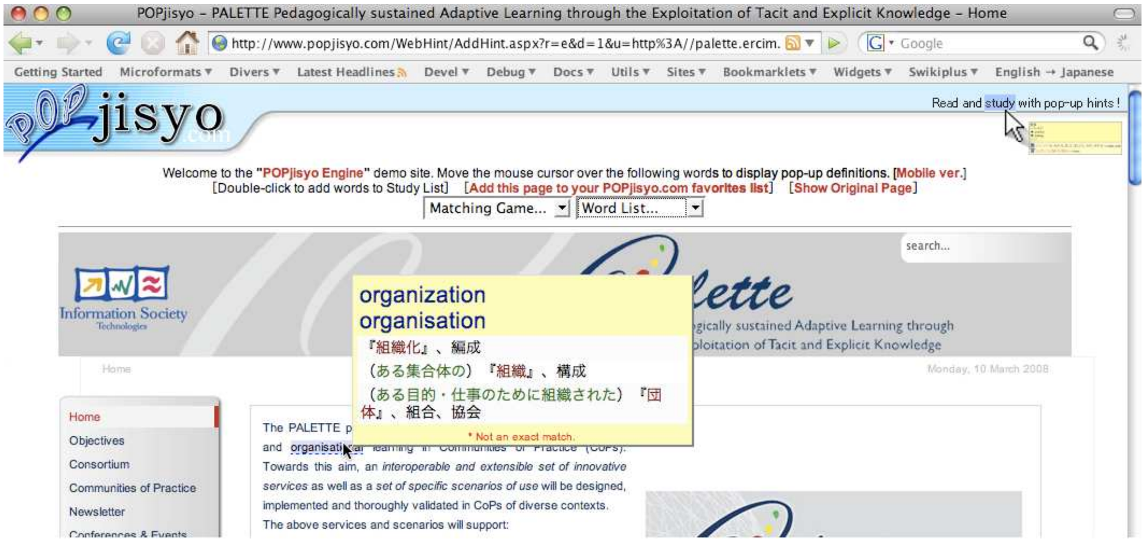
Applicability to PALETTE

The browser extension is well suited to integrate services that were not conceived to be integrated together. It also works without changing the integrated services source code. Thus the browser extension would be particularly useful to integrate legacy services with PALETTE, for instance to extract or to annotate the legacy content.

3.1.9 The bookmarklet

A bookmarklet is a small piece of code self-contained in a URL and that can directly executed by clicking on that URL. The main interest is that a bookmarklet URL can be dragged to the browser's bookmark catalogue or to the bookmark toolbar on the top of the window. Then the user can call it at any time. It will be executed in the context of the page, which is currently displayed in the browser window and have access to its content. As it has a read/write access to the page content, it can dynamically change its visual appearance and thus can perform dynamic composition of services at the presentation layer.

For example, the bookmarklet on the figure below is for exploring a page and getting a translation in Japanese of its words (URL: PPJ). This bookmarklet changes the original page by dynamically inserting a "POP jisyo" header with two popup menus. It also changes the user experience by adding a contextual popup window above English words under the mouse, displaying their translation in Japanese that it gets from an external service.



POPjisyo – PALETTE Pedagogically sustained Adaptive Learning through the Exploitation of Tacit and Explicit Knowledge – Home

http://www.popjisyo.com/WebHint/AddHint.aspx?r=e&d=1&u=http%3A//palette.ercim.

Welcome to the "POPjisyo Engine" demo site. Move the mouse cursor over the following words to display pop-up definitions. [Mobile ver.]
[Double-click to add words to Study List] [Add this page to your POPjisyo.com favorites list] [Show Original Page]

organization
organisation
『組織化』、編成
（ある集合体の）『組織』、構成
（ある目的・仕事のために組織された）『団体』、組合、協会
* Not an exact match.

The PALETTE p… and organisati… Towards this aim, an *interoperable and extensible set of innovative services* as well as a *set of specific scenarios of use* will be designed, implemented and thoroughly validated in CoPs of diverse contexts. The above services and scenarios will support:

Figure 28: A bookmarklet to learn Japanese from any existing Web page, here applied to the PALETTE Web site

A bookmark is very easy to install and installation is instantaneous. It doesn't require a browser restart, as it is the case with Firefox extensions. The bookmarklet is more a technique to extend a service on the fly by injecting code into the client-side of the service components, rather than a visual composition technique, but it can be used for that purpose.

Applicability to PALETTE

The bookmarklet is lightweight to develop and to install. Thus it can be used to quickly prototype and demonstrate some ideas of integration of services. Basically it can be used to augment one service on the fly with contextual data extracted from another service. For instance in PALETTE is could be used to deliver cross-service contextual awareness. For instance, it could be used in services that displays user names, to display in-situ a popup window showing statistics about the last user' actions that it would get from the CAKB.

3.1.10 Summary and raised issues

The visual composition methods offer different solutions to integrate the presentation layer of services. Except for special cases, such as customizable portals, or the browser extensions that aims at improving the user interface of a service, these methods are best exploited when the visually integrated services are also integrated using data composition, service invocation or composition of services as defined in D.IMP.04.

In the table below, we propose three criteria to help developers choose between different visual composition methods:

- "Installation by the user" means that the user can select which services s/he compose together;

- “No need for a mashup host or proxy server” means that the implementation method of the composition will also allow to bypass the same-origin security policy of Web browsers that prevents a service A from calling a service B when executing in the client as defined by RFC3987 (Duerst and Suignard, 2005). This may change in the future as a new proposition is currently under development by the W3C Web APIs Working Group (URL: ACC) that will provide a mechanism to enable client-side cross-site requests.
- “Do not require direct modification of services source code” means that the implementation method of the composition does not require to have access and to modify the source code of one of the services; of course to go beyond merely visual composition this supposes the services offer Web services access.

In the table we have not considered the first, trivial, tab group composition by the browser. Thus tab group composition, like menu bar, means that a service proposes a simulated tab group or a simulated menu bar inside the browser window. We didn’t consider neither the service embedded as an image method, which just requires that the target service is able to produce an image in response to a service call.

	Tab group **Menu bar**	**Action Button Link**	**Widget Portal**	**Mashup**	**Browser Extension**	**Bookmarklet**
Installation by the user			X		X	X
No need for a mashup host or a proxy server	X	X	(1)	(1)	X (2)	Depends (3)
Do not require direct modification of services source code			X (4)	X (4)	X	X

(1) In fact the Portal server and the Mashup host act as a proxy that allows redirecting cross-site requests. Widgets must use special APIs to benefit from this effect.

(2) The browser extension is powerful because, on Firefox, it can do nearly any operation as it is executed with a higher privilege than the basic code, which is executed into a browser window. This allows to easily combining calls to several different services hosted on different domains.

(3) There are some strong limitations to the interoperability allowed between site A and site B through a bookmarklet. This is because of the same-origin security policy of Web browsers that prevent untrusted code to load information across domain boundaries in a Web browser. There are some techniques to go round these difficulties, that would go beyond the scope if this overview. However the most robust solution is to have a part of the bookmarklet code to be delivered through a bookmarklet server host that acts as a mashup host and allows cross-domain service calls.

(4) As long as the target services offer Web services with REST or SOAP APIs. However special effort is required to design and implement the widgets that call these APIs.

The most “X” in a column signifies that the visual composition methods and its implementation offer the most freedom and the least complexity to the user and to the developer.

The visual composition methods all have an implementation cost. In some cases the implementation cost is to directly modify a service, thus posing a new problem which is to decide whether to have two versions of the service or not (i.e. one for running inside the PALETTE environment, the other for running as a standalone application). In other cases the implementation cost is to develop a new application/user interface, such as a widget, a browser extension, a mashup application or a bookmarklet.

In PALETTE, we have decided to privilege the second type of solutions, which are the most interesting according to the analysis of the state of the art summarized in the above table. Among these solutions, and under the auspices of WP5, we have decided to privilege and to support one of them: widgets to be executed in a PALETTE Services Portal. That solution is detailed in the next section. The portal that is being developed in WP5 as a widget container will support visual composition by juxtaposition.

3.2 The PALETTE Services Portal

The PALETTE Services Portal answers to the following CoP's concern: "How to use all of the services from one central place?".It is basically a customisable web portal provided to end-users as a single entry point to PALETTE services. Its customisable aspect stands from the fact it is composed of multiple small embedded applications called widgets, which users can organize the way they want. These widgets can be seen as small windows, offering some function of a service or simply displaying information from a service, and that provide an entry point to the PALETTE services. Users can basically add any number of widgets they want, remove them, personalize them, and organize them in the portal. A tutorial on widget creation and installation in the portal is given in annex 8.4.

The remainder of this section presents the technical aspects of the realised portal as well as some screenshots illustrating the current version. It targets first developers that want to make widgets for their service capable to run in the Palette Portal.

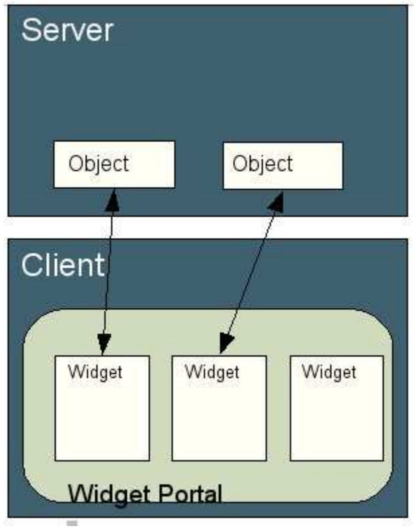


Figure 29: Widgets interacting with server objects

Personalized Web sites are not a novel idea but received increasing attention with the establishment of the Web 2.0 and the Rich Internet Application technologies that allow for much more dynamic and flexible layout and interface composition. While it is not possible to integrate the entire set of services into the presentation layer, the portal may still serve as a single entry point by offering small applications for displaying or updating remote data, while at the same time offering the possibility to access actual services. Such small applications are usually called widgets, defined as being a small embeddable application that can be included and executed within any HTML-based web page. The Web Application Formats Working Group in the W3C Interaction Domain is working on the Widgets 1.0 (URL: W10) specification that is currently released as a public working draft. While the type of widgets addressed by that document are packaged client-side applications that are downloaded and installed on a client machine, the PALETTE portal intends to embed and execute widgets without any download or other user intervention such as compilation.

In order to deliver PALETTE services at the end-user level, the personalized portal displays the widgets of services in the PSRF together with a generated list of links to access the services that don't have any widgets. In our case, a PALETTE widget represents a view of the state of a service, providing access to offered functionalities and an entry point to the service itself. Each service may have several different widgets, where each widget allows interacting with a specific feature of that service (such as displaying information, updating data or interacting with a specific functionality, such as a SOAP or REST Web services).

For the portal to be a customisable homepage with the capability to add PALETTE widgets, it was first necessary to define the specification of the PALETTE Widget Format. This specification can now be used by the different CoPs to create their own set of widgets, which can be integrated and displayed within the portal. Several widget specification or widget formats already exist and have been analysed closely in order to offer the best possible features to the PALETTE Widget developers.

3.2.1 Overview of widget formats

The PALETTE Widget format serves as the development guidelines for the creation of new widgets. While a widget is only a small application, the separation of the data and the user interface is a major concern and decoupling data access and business logic from data presentation and user interaction served as a guideline while designing the widget format. In particular, the architectural Model-View-Controller (MVC) pattern used in software engineering and the associated guidelines were both respected. A detailed analysis of existing solutions and specifications should give enough insight on how these particular problems were solved by other parties.

It should be noted that throughout this document, several different terms are used to describe the same idea. While the majority of the Internet has largely adopted the term widget, the concept of pluggable user interface components is known in the Java world under the name portlet. The decision to employ the term widget rather than gadget, badge, module, capsule, snippet, mini, portlet or flake, which are all names mentioned in the Web widget wikipedia article (URL: WWID), should not give any indication about the technology used but rather the concept of small embeddable applications.

Several parties offer personalized Web sites with their very own set of widgets, developed according to their own widget specification. Among the most popular widget formats are Google Gadgets, Netvibes Univeral Widgets, Yahoo! Widgets, Microsoft Gadgets, Apple Dashboard Widgets, and Opera Widgets. These formats are studied in details in Annex 8.5.

Our research has shown that the analysed widget specifications structure their widgets in at least two different parts, similar to the approach used by the W3C Widgets 1.0 Draft:

- A widget configuration section contains information necessary to initialise and use the widget. This section optionally includes meta-information about the widget and might include the possibility to define the user preferences.
- The main widget file, which is displayed in the widget viewport. This document contains the programming logic, widget layout and structure and is, apart from Yahoo! widgets, static XHTML or HTML that uses JavaScript to access widget-specific functionalities offered by the Widget Scripting Interface, a programming interface to access widget-specific functionalities.

In respect to the architectural Model-View-Controller pattern, we distinguish the following parts:

Model:	this sections contains the widget settings and user preferences
View:	this section contains the widget layout and structure
Controller:	this section contains the programming logic of the widget

3.2.2 PALETTE Widget Specification

Based upon the results of the widget specifications analysis, we now have to define the specification of the PALETTE Widget format. Even though the JSR168 portlet specification (URL: JSR168) could be used, the Java language is too heavy to design a web portal around. Since there is no J2EE architecture that the web portal would need to integrate into, Java had no additional advantages that would make it a better solution than the other formats.

Since the widget formats are all very similar, we decided to use the W3C Widget specification (URL: W10) as a starting point to build the PALETTE format around. The format is simple, flexible and easy to implement. Additionally, the continuous effort by the W3C could make their specification the standard for embeddable web applications that use the browser as runtime environment. There are no advantages to the other widget formats that would justify the need to use another specification.

To respect as closely as possible the specifications and guidelines given by the W3C, we designed the PALETTE Widget format with the intention to be compatible to the W3C Widgets 1.0 specification while at the same time being extensible to offer additional functionalities through the PALETTE Web portal.

Our Widget Specification describes the widget packaging format, the manifest file config.xml and the scripting interface for creating widgets for the PALETTE web portal. 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. The 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.

The first modification was to add the possibility to define the user preferences in the configuration file. This step was necessary to emphasize the Model-View-Controller pattern and to facilitate the development of PALETTE widgets. To stay compatible with the W3C Widgets 1.0 specification, the definition of the user preferences is done in a separate namespace.

The second modification was to distinguish two different types of widgets, called *local* and *remote* widgets. While local widgets are similar to the standard widgets specified by the W3C, Google or Opera, remote widgets are a different type of widget inspired by the resource-oriented approach used in the architecture of the Web (URL: WEBARCH). A remote widget is basically a Web service hosted on a remote server that includes the presentation layer. This style of architecture is similar to the Representational State Transfer (REST) model introduced by Roy Fielding (URL: FLD), as it describes an interface that transmits data over HTTP without any additional messaging layer. A remote PALETTE Widget is a resource, identified by a global identifier (its URI), and the interaction with these resources is done via a standardized interface (*e.g.* HTTP) to exchange representations of these resources (URL: REST). The representation of the resource, in the case of the PALETTE portal, includes the presentation layer that can be used by the end-user to interact with the resource.

While local widgets are static client-side applications implemented in HTML and JavaScript, remote widgets may include server-side scripting languages that dynamically produce the widget content. In theory, while it is possible to implement any type of widget using both approaches, it facilitates the transformation of existing PALETTE services into widgets. In fact, any URI can be transformed into a remote widget and integrated into the portal, making the development of new widgets as easy and fast as possible. It should be noted that remote widgets access user preferences through GET parameters and for security reasons and technical limitations, have no possibility to use any Widget Scripting Interface within their JavaScript code.

Following our analysis and observations, the PALETTE Widgets specification has been formulated, which can be found in annex 8.1.

3.2.3 PALETTE Portal design and implementation

In this part we describe the main points regarding the design and implementation of the first version of the PALETTE Services Portal.

Analysis

Actors

The actors of the system are the following:

Users: Everyone with access to the portal is a user. A user has the possibility to customize its interface by adding, modifying or removing widgets and to personalize the content of the widgets by modify its preferences.

CoP Service Delivery agents: Member of a CoP that has special access rights to the management interface of the CoP widgets. It is probably the CoP mediator.

Administrator: The administrator's task is to manage the users and promote users to CoP Service Delivery agents. It can be the CoP mediator or the software administrator of the CoP.

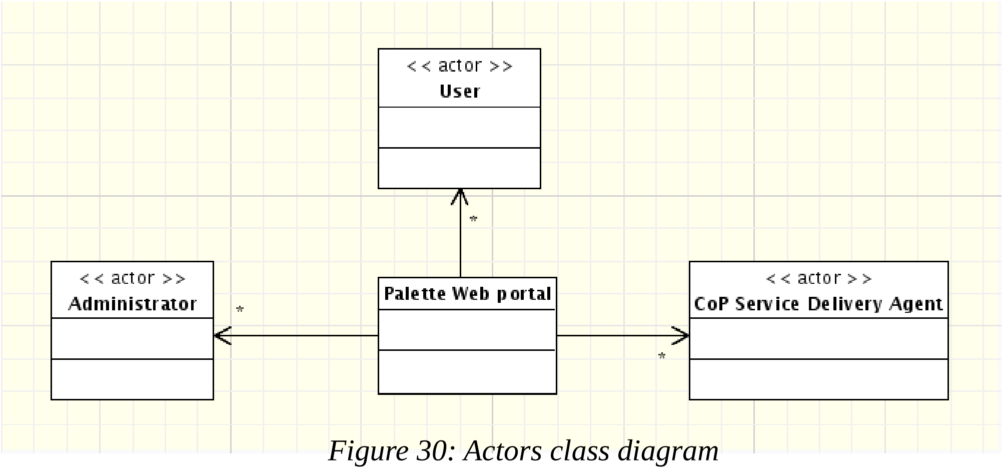
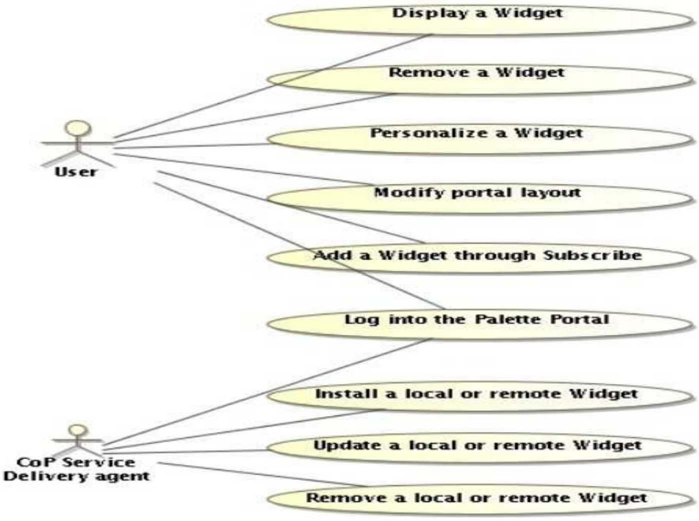


Figure 30: Actors class diagram

Use cases

Here is the list of basic use cases identified for the portal:



Display a Widget
Remove a Widget
Personalize a Widget
Modify portal layout
Add a Widget through Subscribe
Log into the Palette Portal
Install a local or remote Widget
Update a local or remote Widget
Remove a local or remote Widget
User
CoP Service Delivery agent

Figure 31: Use case diagram

Design

Following our analysis of the requirements, we can now start modelling the web Portal. As we are in a web development field, we decided to use a standard three-tier client-server architecture. The separation of the user interface, the functional process logic and the data access/storage allows us to develop each part or tier as an independent module and to update or replace that module in the future.

The definition of each tier in the case of the PALETTE portal differs from the “classic” three-tier architecture for web development, where we have the Presentation tier, the Application tier and the Data tier. In fact, the Rich Internet Application (RIA) technologies allow us to extend the client with parts of the process logic and distribute the load off the server. This technique has been used during the implementation of the portal, as shown by the following example illustrated in Figure 32.

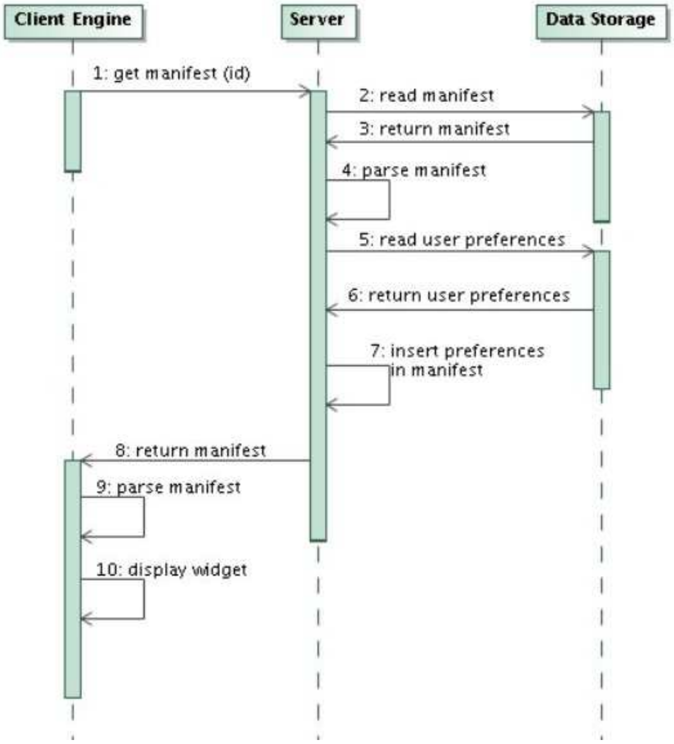


Figure 32: Sequence diagram of a widget display

As shown by the sequence diagram, displaying a widget within the interface goes through several different phases, with parts of the processing done on the server as well as on the client. The process is as follows:

1. The client receives the user requests and sends a request to the server to receive the widget manifest. Since the request is asynchronous, the client continues to operate normally.
2. The server receives the request. The server uses the id from the request to read, parse and validate (validation can be disabled to increase performance) the widget manifest from the file system.
3. The server accesses the database and reads the user preferences for this widget. It then inserts the preferences in the manifest and returns it to the client.
4. The client receives the manifest from the server. It parses the manifest and constructs a widget object. The widget information are loaded and inserted in the user interface.

The above scenario is actually slightly more complicated if we assume that the user request for a certain widget needs to be validated first by the server or if the widget is a remote widget, in which case the client needs to access the remote server too. In the following chapters, we present in more detail each tier and their implementation.

Storage Design

The PALETTE web portal requires two types of storage, a relational database and the file system. The need for the file system was introduced by the possibility to upload local widgets, which essentially are archives that may contain all kind of files, such as images, JavaScript, CSS or HTML files. The relational database is used for all the other data, such as users, CoPs, widgets, preferences or tags. We decided to duplicate parts of the information from the manifest stored on the file system within the database, which allows us to use the processing power of the DBMS to formulate advanced requests that would not have been possible using only the file system.

The relation database stores the information that is necessary for the PALETTE portal to operate correctly. The problem domain of the PALETTE portal is as follows:

The PALETTE portal has a certain number of users, each with a unique identifier, specifically their username, and a password to log in the portal. While it is not necessary at this time to store further information about the user, such as his email or real name, the database design should be flexible enough to account for such a requirement in the future. The PALETTE portal also has a certain number of Communities of Practices, identified by their CoP name. Similar to the users, we should be able to add more information to each CoP in the future. We also want to store information about the widgets in the database. A widget belongs to the CoP that has deployed it on the server. It should be possible to assign a widget to a certain category, make it visible/invisible to normal portal users and to associate a number of tags to the widget.

Each user has his personal interface, in which he can display widgets. The displayed widgets can be situated in one of the interface columns, at a certain position (the position is determined by the order of display of the widgets in a column) and it can be displayed in a minimized or maximized state. Additionally, each widget can be customized through a number of preferences. A preference is determined by a key – value pair and is specific to a certain user.

Finally, users may have special administration rights that allow them to organize the widgets of the CoP they belong to.

A detailed analysis of the above requirements allowed us to create a logical data model of the problem domain.

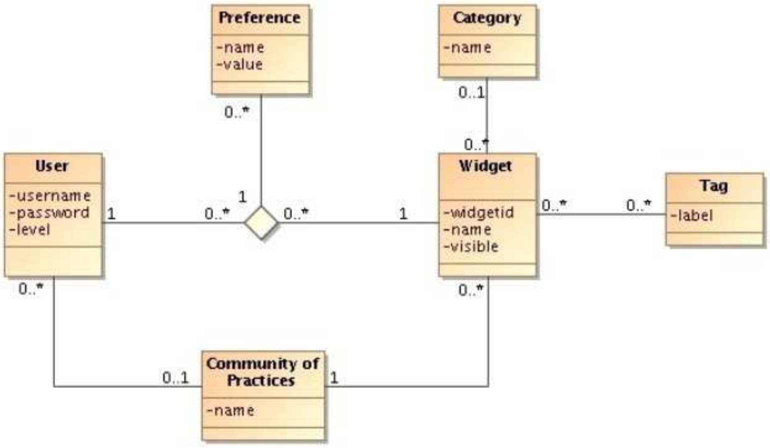


Figure 33: logical data model (UML)

Taking into account the facilities and constraints of our database management system, which was MySQL using the InnoDB storage engine with foreign key support, we derived the physical data model.

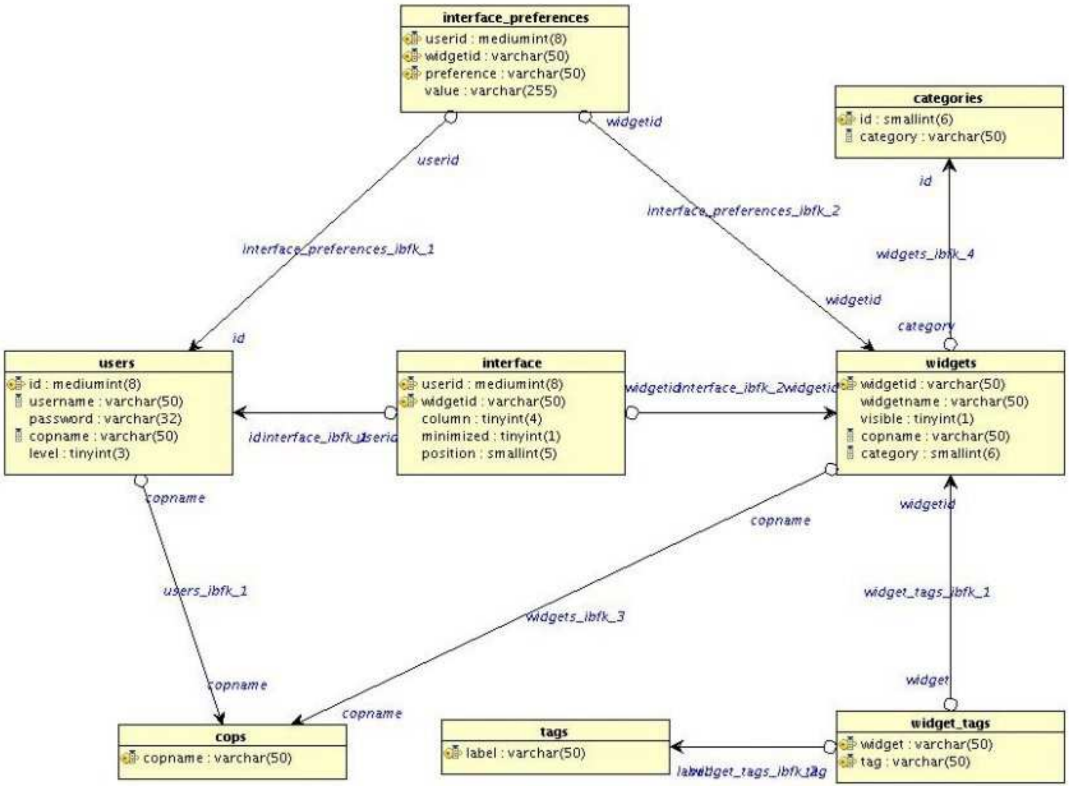


Figure 34: Physical Data Model

<u>File System Design</u>

The file system is used to store the uploaded widgets, whether they are local or remote. For this purpose, within the deployment folder on the web server is a special directory, to which the web server service has write permissions. If a new widget is uploaded, it first has to go through a validation phase, during which the portal verifies if the manifest is valid according to the XML schemas, the widget specification, such as packaging, have been respected and widget id, specified in the manifest, is unique. This id is then used to generate a new widget directory within the writable directory on the server.

In the case of a remote widget, only the manifest is copied into the new directory. In the case of a local widget, the entire archive is verified and extracted into the directory. To protect the file system against malicious code, such as server-side scripts that would attempt to delete parts of the file system's content, a special directory-level configuration file, also known as Apache's .htaccess file, is used to disable server-side script interpretation within the widget directories. If a widget requests a script that is located in a widget directory, this script will not be interpreted but returned as simple text.

<u>Server Design</u>

In the server design, two different parts can be distinguished: the scripts and services that are used internally by the portal itself and the REST web Services that open up the functionalities of the entire portal to the outside. The following two sections explain in greater details each of these parts.

Web Portal

The decision to use Ajax as a Rich Internet Application technology has an important impact on the design of the server. Contrary to classic web development where dynamic server-side scripts generate static markup, most of the time already containing the layout of the website, the Ajax approach is to transmit data between the server and the client engine. As such, the server scripts have been designed as small web services. They do not support as many operations as the web services in the next chapter but they are used to receive parameters through a GET request and return either XML or JSON encoded data back to the client. The transformation from the data to XHTML is done in the client engine, which makes the Server design a lot easier than classic projects without Ajax: the web services are small, light-weight standalone scripts that only depend on a common authentication object and a configuration file.

The different services integrated in the portal are the following:

- a category service to read the currently available categories in the database
- a manifest service to read the manifest of a given widget, including the user preferences
- an interface service to read the widget positions and states of a given user
- a tag service used for tag auto-completion, read the tags of a given widget and read the most used tags
- a widget service to read the currently available widgets in the database
- an interface service to upgrade the widget positions and states of a given user
- a preference service to update the preferences of a given user and a given widget

The access to the Storage database is using a special database abstraction layer, namely Pear MDB2. This Pear package, released under the BSD License, provides a common API for all supported Relational Database Management Systems. It provides most of its features optionally to ensure the construction of portable SQL statements. The advantage of using MDB2 is that it is possible to change the used RDBMS in the future and to have it work automatically without the need to change a single line of code. The only noticeable difference is that MDB2 (URL: MDB2) is using an appropriate driver (that needs to be installed on the server) and the new Data Source Name.

The access to the Storage file system is done using the standard PHP File System functions. These functions guarantee an interoperable access mode for every operating system.

The last feature worth mentioning of the web portal is the content proxy. Because of the security restrictions of the XMLHttpRequest object (URL: XHR) used by Ajax, which only allow to access files on the same server, we had to create a possibility for the different widgets to requests scripts on other servers. Our solution is a simple web proxy script that passes all requests and replies unmodified from the source and the target. The correct header values of the reply are preserved in order to give the client engine the possibility to parse the reply with the correct method automatically. The content proxy is implemented using the Pear HTTP-Client package (URL: HTTPC). This package provides a higher level interface to perform multiple HTTP requests. Since the PHP functions are not subject to the same restrictions imposed on the XHR object, we are able to load the content of remote scripts and return it transparently to the XHR object.

Web Services

The second major part of the server design is the REST web services. It is a software system designed to support interoperable machine-to-machine interaction over a network. The PALETTE web services offer programmatic access to the services of the portal and allow developers to access these services using their preferred programming language. In addition, the use of standards-based formats and communication methods, such as XML and HTTP, make web services platform-independent. They expose the business logic of the portal to the Web, increasing its usability.

We have implemented restful web services on the portal, but since their interface is not yet stabilized, we will not describe them further here.

Client Design

Introduction

As mentioned in the introduction to the PALETTE web portal design, the client has been written using the Ajax approach. It has a powerful object-oriented client engine written in JavaScript and uses a complete OO approach to overcome the complexity of the processing logic in the client. Because of its asynchronous nature, the user can continue operating the interface while the client engine is loading new widgets or updating the user preferences. This is especially useful when the user connects to its customized portal and several widgets needs to be displayed at the same time. If we had used a synchronous approach, the widgets would have loaded one after the other, instead of all at the same time.

Contrary to the rather simplistic implementation used for the Server tier, the client required a more structured approach, such as a class diagram, and the asynchronous nature posed additional challenges that had to be overcome. Before presenting the implementation of the client, we'd like to introduce the current features of the client.

Features

Apart from the required functionalities in the client engine, the client has lots of interesting features that facilitate the development of new widgets and the interoperability with the portal.

Local Development Environment:

As developers, we understand the necessity to have a powerful development environment, which allows visualising the result without much hassle. Especially for the development of local widgets, we realised that it was too much work to archive the files, deploy the widget within the portal and verify the correct functioning of the widget, each time an update was done. That's why we designed a Local Development Environment. By simply including a JavaScript file and a CSS file hosted on the PALETTE portal, a normal HTML file could be turned into a widget, giving it all the properties and features that it would have in the portal. The JavaScript files includes the complete manifest parser, the widget object defined in the widget specifications and using the provided stylesheets, transforms the content of the page into a widget, using the look&feel of the portal.

This approach allows widget developers to test their widgets without any additional effort on their machine and provides an efficient development environment that doesn't require any deployment on the portal.

Widget Subscription:

One of the problems of Ajax is that the source of the dynamic content that is loaded asynchronously into the page is hidden to the user and it is not possible to reference it through an URI from other websites. Links and interconnected resources have been one of the fundamental design rules of the Web and the Widget Subscription option provides this functionality even for the Ajax-driven portal. The subscription feature is a special URL that takes a widget id as argument and automatically adds that widget to the user's interface. If the user is not authenticated (no existing login session available), the user is first requested to login before the widget is added to his interface.

This feature allows Service providers or Widget developers to create a list of interesting widgets on their respective websites, maybe with a detailed description of the functionalities of the widget, and offer a One-Click solution to the visitors the add the desired widget to their interface.

External Widget Integration:

Providing CoP users with a single entry point to the PALETTE services was not enough, that's why we decided to offer the possibility to display a user's personalized widget in other websites as well. A special server script uses the existing user session to display the widget specified as argument, with the same look and feel but without the customisation options (such as remove, collapse or edit). The recommended way of using this feature is to create an iframe in the website that wants to display the user's widget and use the provided URL as source of this iframe. The widget will display and integrate perfectly into the website.

Widget Management Interface:

The Widget Management Interface is a restricted area of the portal, where only CoP widget managers with the correct user permissions have access to. This Ajax-powered interface allows them to comfortably deploy new widgets (upload and install), edit existing widgets or remove them completely. Additionally, they have the possibility to make widgets visible/invisible to the normal users. Invisible widgets are not possible to be added to a user's interface and they do not show up in any of the widget lists on the portal. This feature allows widget developers to first verify the correct deployment of their widgets on the portal and only make them accessible if the widget has been carefully tested.

The Widget Management Interface also provides the managers with the needed URLs to use the Subscription and External Widget Integration feature.

Widget Tags:

Collaborative annotation of the portal widgets is supported through a tagging system. Each user has the possibility to add tags to a widget in the respective preferences window. These tags can then be exploited in numerous ways, one of which is the Tag Cloud Widget, that display the most used tags and the widgets that have been annotated with. This allows users to quickly find interesting widgets and add them to their interface.

Design

JavaScript, the programming language used in Ajax, has powerful options for Object-Oriented programming. The common OOP analysis approach allowed us to create a detailed class diagram first and derive the implementation from it.

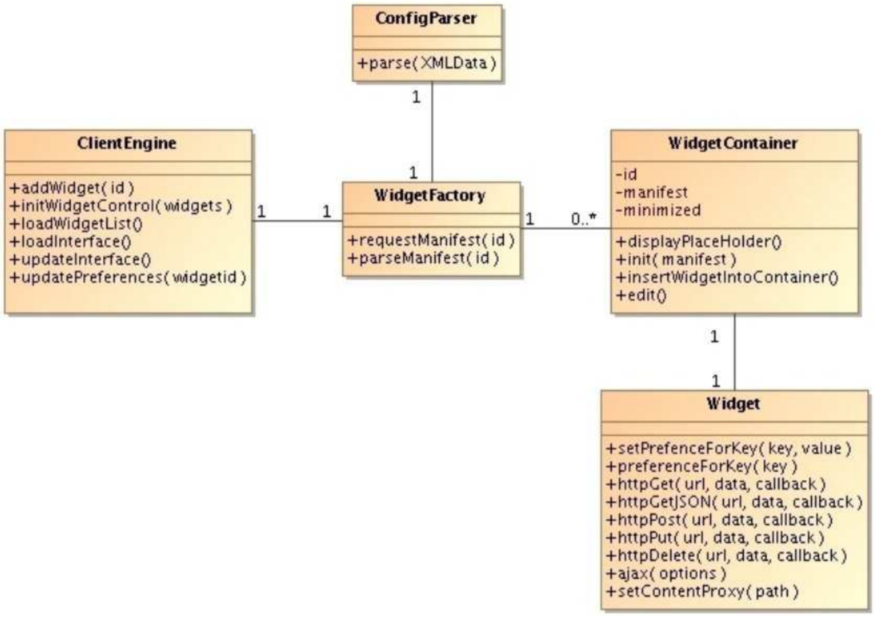


Figure 35: Class Diagram of the Client Engine

The client is made up of several classes, represented in the class diagram above. Here's a short description of each of the different classes:

- ClientEngine: handles most of the asynchronous server calls and is responsible to synchronize the client with the server

- WidgetFactory: organizes the client widgets. Requests new manifests from the server, controls their parsing and initialises new WidgetContainers.
- ConfigParser: parses and validates the widget manifests and returns a multi-dimensional arrays containing their values.
- WidgetContainer: controls the behaviour and look&feel of the widgets in the portal. Initialises each widget component and generates the widget visualization as well as the edit window.
- Widget: contains all the methods available to the JavaScript of a local widget.

Implementation

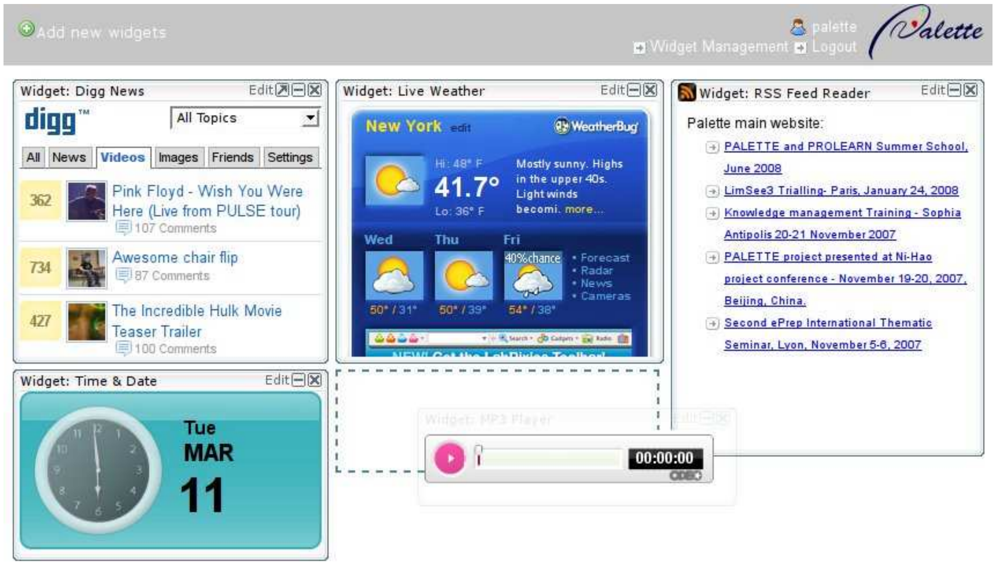
The implementation of the client engine is done in JavaScript, using the jQuery library (URL: JQ). jQuery is a lightweight web application framework that emphasizes the interaction between JavaScript and HTML. It is released and maintained by John Resig, a JavaScript Evangelist, working for the Mozilla Corporation, and co-designer of the FUEL JavaScript library (included in Firefox 3). jQuery offers many interesting features, such as DOM traversal and modification, Events, Effects and animations and XHR wrapping methods. It is mainly used in the portal for unobtrusive event management, powerful DOM traversal using CSS selectors and object animations.

The Drag&Drop support has been implemented using a jQuery plugin called jQuery interface (URL: JQI), a collection of rich interface components. It is released and maintained by Stefan Petre and Paul Bakaus, one of the jQuery core developers. The effect has been created using the sortables plugin, which requires the draggables and droppables plugins. It makes it possible to move elements from one container to another.

The communication between the client and the server uses JSON (URL: JSON) to transmit the data. JSON (JavaScript Object Notation) is a text-based, human-readable format for representing objects and other data structures Contrary to XML, which is a markup language, JSON is a data interchange format, which makes it less complex and easier to use. Since JSON s a subset of JavaScript, it can be used without any additional problems.

3.2.4 Current version screenshots

The following is a screenshot of the latest version available at the time of this writing.



362 Pink Floyd - Wish You Were Here (Live from PULSE tour) 107 Comments

734 Awesome chair flip 87 Comments

427 The Incredible Hulk Movie Teaser Trailer 100 Comments

New York edit WeatherBug

Hi: 48° F 41.7° Lo: 36° F

Mostly sunny. Highs in the upper 40s. Light winds becomi. more...

Wed 50° / 31°
Thu 50° / 39°
Fri 40%chance 54° / 38°

- Forecast
- Radar
- News
- Cameras

Palette main website:

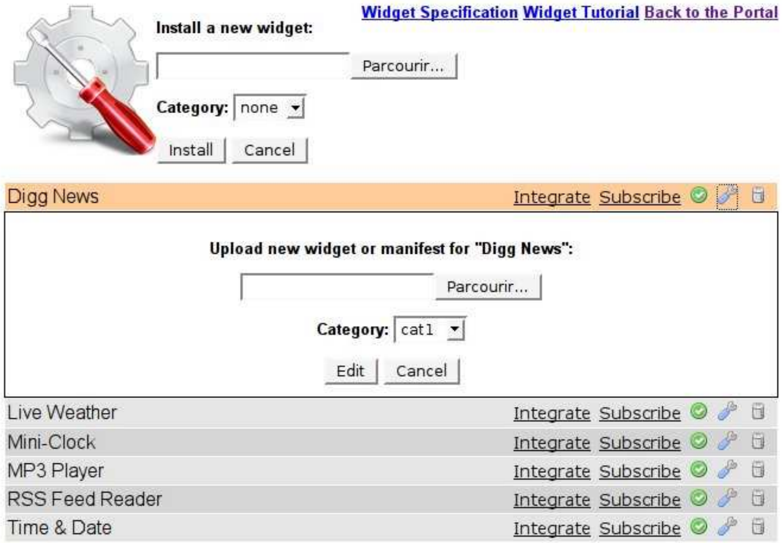
- PALETTE and PROLEARN Summer School, June 2008
- LimSee3 Trialling- Paris, January 24, 2008
- Knowledge management Training - Sophia Antipolis 20-21 November 2007
- PALETTE project presented at Ni-Hao project conference - November 19-20, 2007, Beijing, China.
- Second ePrep International Thematic Seminar, Lyon, November 5-6, 2007

Tue MAR 11

Figure 36: Screenshot of a user's interface

Shown on the previous screenshot are several widgets: three are displayed in their normal state, while one widget, a Flash MP3 Player, is currently being dragged to the middle column, and the RSS Feed Reader widget is in the "Edit Preferences" state. Ajax allows complete manipulation of the portal interface without the need to refresh the entire website. The transition between different states is seamless and provides a satisfying user experience.

The widget management interface regroups all the existing functionalities into one convenient interface. The different options give access to the respective forms, blended in dynamically using the jQuery animations. Users are informed through clean and concise error messages of the result of their operation and, if necessary, the reason of failure.



Install a new widget:

Parcourir...

Category: none

Install Cancel

Digg News — Integrate Subscribe

Upload new widget or manifest for "Digg News":

Parcourir...

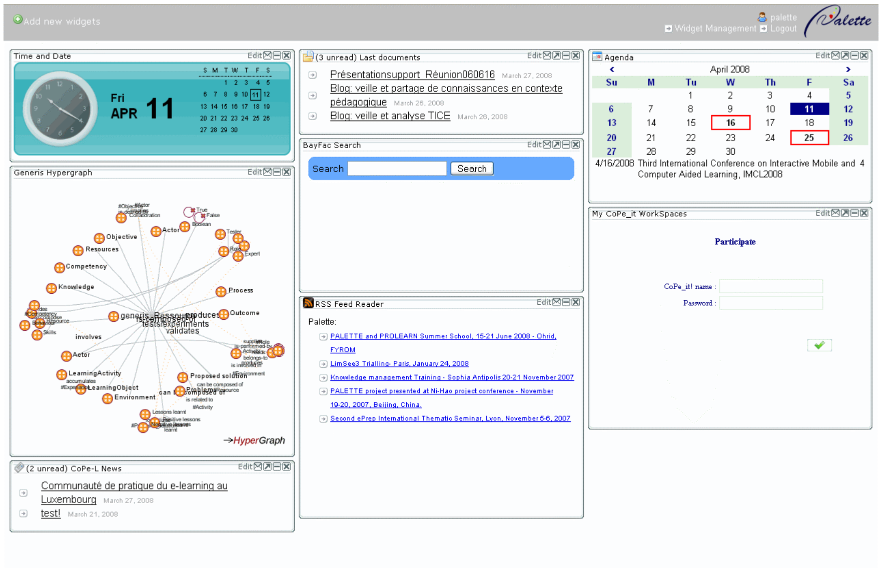
Category: cat1

Edit Cancel

Live Weather	Integrate	Subscribe
Mini-Clock	Integrate	Subscribe
MP3 Player	Integrate	Subscribe
RSS Feed Reader	Integrate	Subscribe
Time & Date	Integrate	Subscribe

Figure 37: Screenshot of the Widget Management Interface

The last screenshot on Figure 38, illustrates the instanciation of the PSP for a CoP. In this case, this was the CoPe-L. Here, widgets providing views to PALETTE services (BayFaC, CoPe_it) are mixed with others: a simple RSS feed reader pointing on PALETTE's web site, the news of the CoP, a cealendar, etc.



Time and Date

Generis Hypergraph

(3 unread) Last documents

- Présentationsupport_Réunion060616 March 27, 2008
- Blog: veille et partage de connaissances en contexte pédagogique March 26, 2008
- Blog: veille et analyse TICE March 26, 2008

BayFac Search

Search [ ] Search

RSS Feed Reader

Palette:

- PALETTE and PROLEARN Summer School, 15-21 June 2008 - Ohrid, FYROM
- LimSee3 Trialling- Paris, January 24, 2008
- Knowledge management Training - Sophia Antipolis 20-21 November 2007
- PALETTE project presented at Ni-Hao project conference - November 19-20, 2007, Beijing, China.
- Second ePrep International Thematic Seminar, Lyon, November 5-6, 2007

Agenda

April 2008

Su	M	Tu	W	Th	F	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

4/16/2008 Third International Conference on Interactive Mobile and Computer Aided Learning, IMCL2008 4

My CoPe_it WorkSpaces

Participate

CoPe_it! name :

Password :

(2 unread) CoPe-L News

- Communauté de pratique du e-learning au Luxembourg March 27, 2008
- test! March 21, 2008

Figure 38: PSP for the CoPe-L

4 The Cross Awareness Knowledge Base

4.1 Introduction

The Cross Awareness Knowledge Base (CAKB) answers the following CoP's concern: "How to have a global view of knowledge and activities in the CoP?" It simply provides a mean to have in a single web page information coming from all services used in the CoP.

According to D.PAR.04, the current PALETTE services do not cover some CoPs' concerns: "When using most of these tools, some questions remain unanswered: Who are the members of a CoP? How do you access the personal information of a member (email, telephone number)? When does a certain event take place? Who is present when, and where are they? Who is interested in what subject? What content do we have access to and where is it located? Yet access to information about the activities of the CoP seems essential to keep it alive and thriving." Indeed, those questions concern one single aspect: awareness. Awareness of what happends in a CoP's day-to-day life, and awareness of CoPs' members and their profiles. The CAKB has been designed to answer the first kind.

The initial objective [2] of the CAKB was to "give cross awareness of activities within the CoP, an overview on all resources (pieces of information: data and meta-data) whatever their location, whatever which tool manages the content." The CAKB provides a service for users and services to search and access the knowledge managed by the PALETTE services. From this first view and after discussion with the WP5 partners, the functions of the CAKB have been divided into four steps described in section 4.3: PALETTE services knowledge capitalization, knowledge structure, knowledge access to users and knowledge access to PALETTE services. These functions are illustrated by the use cases presented in section 4.2.

4.2 Use cases

The following use cases illustrate the functions of the CAKB. The first part describes use-cases illustrating the main function of the CAKB, which is action awareness while the second part presents other use cases that will be investigated.

4.2.1 Main use cases based on action awareness

The main function of the CAKB is currently to retrieve and store any action performed in registered services, making it available for users' and services' awareness. The two following use cases are examples of what can be done with such function.

Awareness for conflict avoidance

As a same document can be used by different PALETTE services for different purposes, there can be duplicates and different versions present in the CoP repositories. This can become a problem when a member, author of a document, wants to make some changes for example because of a mistake she wants to correct.

When a CoP member makes such a modification, the CAKB is made aware of this modification. Then, because the CAKB is aware of the existing duplicates of the concerned document and the different existing versions, it can warn the CoP administrator or the services storing the different versions so that some homogenisation action can be taken. As soon as the CoP administrator has the correct access rights on the different services, and because he can access all the needed informations from the CAKB, he can decide what to do.

Knowing the value of knowledge

To determine the value of documents as knowledge for a CoP, one needs to know how they are used, how many people have consulted them, what kind of people (expert, curious…) they are. It is a

[2] Palette Developers' Meeting held in Patras, Greece, on the 26-28/02/2007.

statistical work taking into account members participation and their interactions with the available documents. To perform this work across each service of the CoP, one needs the information capitalized by the CAKB about the actions performed on resources. The ontology of the CAKB described in section 4.4 has the elements needed to perform these statistics in an automatic way in the future knowledge evaluation framework to be presented in D.KNO.07.

4.2.2 Investigation use cases

The CAKB can provide more than simple action awareness, as we show here in two additional use cases. While being technically feasible, the implementation of such use cases would however require additional development of PALETTE services. Then, because the feasibility of such a development still remains to be discussed, we propose these use cases as perspectives, illustrating what could be achieved.

Inter-service synchronisation

CoP members would like to synchronise the documents managed by two services they use. Thus, the services developers (or mediators) have to deploy a routine that sends queries to the CAKB, asking the pathway to reach documents managed by the other service (see section 4.3.4). The result is a list of document managed by the other service, and the pathway to reach each of them. If the service does not yet manage one of them, the routine can import this document via the pathway indicated in the CAKB response or simply make a link.

Cross-service search

Members working on a collaborative service like eLogBook or CoPeIt!, want to add documents available on other web applications, according to given parameters such as: a match with the current activity, have been written by some specific person, etc. With a plug-in sending a request to the CAKB, the collaboration service can be given the list of matching documents of all applications that actually send information to the CAKB. Such plug-in could be presented as a widget with the available documents in a combo-box (in this case, a contextual query has already been sent) or with one or two combo-boxes to decompose the query to send (service choice, subjects choice…). In this use case, the CAKB avoids having to send explicitly a request to each service or application used by a CoP: it serves as a central point of information.

4.3 Functional specifications

In this part, we describe the current functions of the CAKB.

4.3.1 Knowledge capitalization

The first step is to populate the CAKB with the knowledge managed by the different services. These latter have to choose which of their knowledge is relevant for the CAKB. Knowledge gathering will be performed according to one or multiple of the above methods:

- The CAKB can subscribe to different PALETTE services RSS feeds. In these RSS feeds, each service traces the actions (Create, Read, Update or Delete) executed on their resources.
- For these RSS, the services having traced new actions can send ping notification signal[3] to the CAKB in order to ask it to load the new RSS. If the service does not send notification to the CAKB, the latter checks for changes in the RSS on the regular basis.
- The CAKB will also be able to call the web services provided by PALETTE services, to get the same type of knowledge.
- Stand-alone applications should be able to send information or updates to the CAKB by calling the web services of the CAKB.

[3] http://en.wikipedia.org/wiki/Ping_blog

4.3.2 Knowledge structure

The second step is to store the gathered knowledge in an efficient manner. The CAKB is a registry for resources and their annotations, present in the various PALETTE services of a CoP. As a registry, it stores only links to resources and their annotations. Aiming to record information about the actions undergone by CoPs members' resources on the different PALETTE services, these pieces of information could be about members, documents, activities, or anything relevant for a global view at the CAKB level.

For this purpose, we have developed an ontology that is used for:

- The storage of resources hyperlinks (doc, person description…) on PALETTE services.
- The storage of metadata (on linked resources) provided by different PALETTE services.
- The storage of actions executed on resources to enable history and statistics on that knowledge.

4.3.3 Consultation

Once populated, the CAKB can be consulted via two Web interfaces. The first one allows CoPs members to consult the resources currently managed by PALETTE services. The second one is for the consultation of the actions (Create, Read, Update or Delete) done on the CoP's resources. This will permit further statistical works about knowledge circulation or lifetime. These interfaces are more detailed in the section 4.6.2.

4.3.4 Inter-services communication layer

It could be interesting to enable the CAKB to provide CoP services with a pathway to make requests to other services. This means that a service will be able to ask the CAKB a pathway to reach resources matching some criteria and managed by any service.

This communication layer of the CAKB is a proposition to the inter-services communication problem, dealing with awareness at a knowledge level. In both solutions proposed below, the communication is established from information captured by the CAKB.

- 1st solution: the storage of pathways in the CAKB for inter-services communication. This aims to provide a way, e.g. a service to call or an URI, for the communication with another service. The service can then automatically make a point-to-point communication bridge.
    - Requests of a service could be like: give me the pathway to get resources with metadata matching "author" containing "Robert".
    - Responses could be (syntax not defined yet) like:
        - url_to_call: array(uri1,uri2…);
        - web_service_to_call: array(call1, call2,…)
    - The service could then reach the searched resources with the returned pathways.
- 2nd solution: the CAKB constitutes a bridge for knowledge transfer.
    - In the same way, the service sends a request to the CAKB to reach some particular resources.
    - The subject of the request is no more the pathway to reach the searched resources, but directly the resources themselves.
    - The CAKB proceeds then the request and sends the list of matching resources to the calling service.
    - This solution has an important inconvenient. The CAKB becomes a single point of failure and, thus, a bottleneck in the inter-service communication.

The implementation of such inter-services communication requires some developments from each service to the CAKB. The later playing thus a central point role, this avoids service-to-service development and moreover allows each single service to reach knowledge managed by any other service without having to know which service in particular to contact.

4.4 Ontology

In this part, we introduce the ontology built for the CAKB to describe the services resources and their actions. Its conceptual graph is given in Figure 39. The RDFS files and its description can be found in annexes.

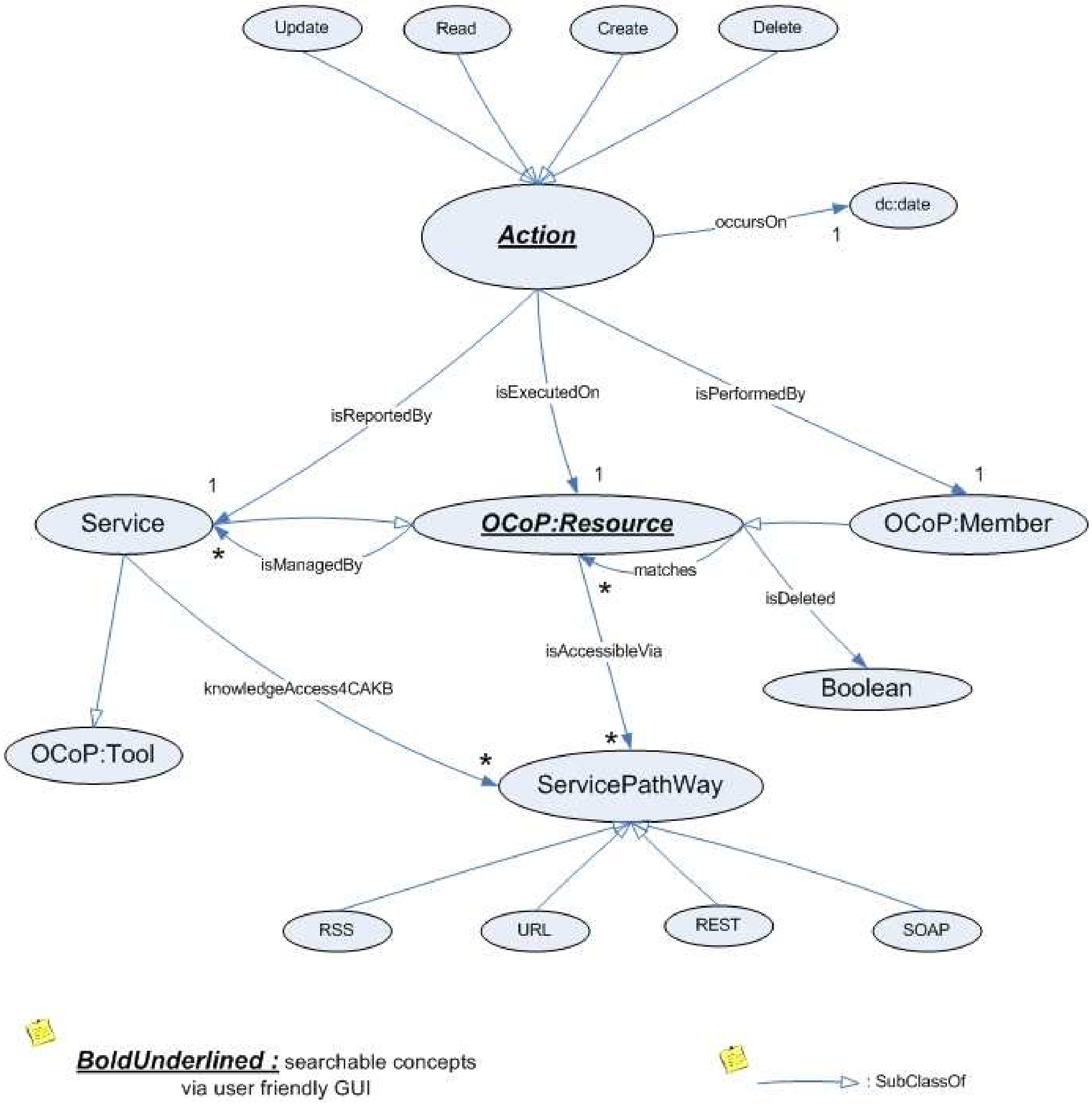


Figure 39: Conceptual graph of the CAKB ontology

In order to provide the functions we have described in a previous section, the CAKB needs first to record the flow of knowledge managed by different services of a CoP. It must also specify the information to access a service, and more particularly to its resources. We choose to store this information in the form of actions and resources. The actions storage allows keeping traces of the knowledge used across the different services and performed by some members.

A central concept is the “**Action**” one. Instances of this concept will be accessible via a search interface that will trace a history of actions on ICT tools of a CoP. The four possible actions (Create, Read, Update and Delete) are subclass of this “**Action**” class to specify the related action type.

An action:

- is an instance of the “**Action**” class,
- is reported (“**isReportedBy**” property) by a service (“**Service**” class),
- is executed (“**isExecutedOn**” property) on a resource ("**ocop:Resource**” class),
- is performed (“**isPerformedBy**” property) by a member (“**Member”** class) of the CoP, and
- occurs on (“**occursOn**” property) a particular date (“**dc:date”** (URL: DCMI)).

The other central concept is “**ocop:Resource**”. Instances of this concept will also be accessible via a search interface allowing viewing the resources of a CoP currently available on its various ICT tools.

A resource:

- is an instance of the “**ocop:resource**” class,
- is managed (“**isManagedBy**” property) by a service (“**Service**” class),
- matches (“**matches**” property) to other resources. In fact, these are the same resources but managed by different services. This property will serve to improve the coherency between the same resources duplicated on different services,
- is accessible (“**isAccessibleVia**” property) via a path (“**ServicePathWay**” class) to a service,
- and, is perhaps deleted (“**isDeleted**” boolean property).

The “**ServicePathWay**” concept enables to store the access way to a resource (“**isAccessibleVia”** property). For the different types of access, a subclass is provided. Thus, there are concepts of “**URL**”, “**SOAP**”, “**RSS**” and “**REST**” as specialization of the “**ServicePathWay**” one. The properties of these classes will be added to the ontology as and when required to provide precise technical information allowing to actually create communication bridges between services. As a first step, the property “**rdf:value**” may contain the code string to execute (the concerned URL or the web service to call…) by the service wanting to access the concerned resource.

4.5 RSS

In this section, we describe the RSS feed that the KM PALETTE services will have to provide. It contains classically elements for describing the channel and the items it includes. An example XML/RSS file is provided in annexes. The chosen format is an extension of the RSS 1.0 standard (URL: RDFSS). We choose this RSS standard because it is written according the semantic web formalism. It enables to add some RDFS (URL: RDFS) elements. It’s useful in our case since we populate the CAKB ontology from this RSS. The extended elements of the RSS have the namespace of the CAKB.

4.5.1 Channel description

• rdf:about	URL of this RSS.
• rss:title	Title of this RSS (not really important in our case but still mandatory)
• rss:link	URL of this RSS (again but mandatory)
• rss:description	Quick description of this feed.
• cakb:paletteService	Label of the concerned service (or the id when the PSRF will be used)
• rss:items	Items list.

4.5.2 Items description

Each item describes an action occurring in the concerned service. A first part is common for every action type. This common part is the only one for “Delete” and “Read” action types.

Common part

- rdf:about Unique identifier of this action. The generation of a unique identifier is in charge of each service.
- rss:link Hyperlink to the concerned resource (which has undergone the action)
- rss:description Quick description of this item.
- dc:date Occurring date of this action
- dc:type Action type : “Create”, “Read”, “Update” or “Delete”.

- ocop :member Description of the member who has performed this action
  - dc:identifier Member URI (or unique id)
  - dc:title Member name

- cakb:isExecutedOn Resource having undergone the action
  - dc:identifier URI of this resource
  - dc:title Label of this resource

Creation and update part

For the description of an action concerning a resource creation or update, the access path and all available annotations are required in the “cakb:isExecutedOn” section of the RSS. The provided annotations can then be displayed in the consultation interface.

- dc:type Type of this resource (“paletteResource” as default value)
- cakb:isAccessibleVia Description of the access path to this resource
  - dc:type Access type (URL, SOAP, REST, RSS)
  - dc:source Description of the access according to its type. For URL access, it is the concerned URL. The other cases are not managed for the moment.
- For each literal property (“rdf:label” for example)
  - A Cdata section with the literal
- For each conceptual property (“ns:isWrittenBy” for example)
  - dc:identifier URI of this resource
  - dc:title Label of this resource

4.5.3 Authentication

Since services manage private resources, the CAKB asking for an RSS service, will also need to send authentication parameters. We proposed that services use a simple mechanism, which is a basic HTTP authentication, described in the RFC2617 (URL: HTTPA). It is not really a secure mechanism but could be enough when it is coupled with the HTTPS protocol.

In short, the login and the password are encoded in base 64 and written in the HTTPS request header. The service will write its RSS feed according to the corresponding rights.

4.6 Current state

This part describes the current state of the development effort. The current test prototype of CAKB is available at (URL: CAKBT). Most of the specifications are met in a simple way as a proof of concept. Last developments are in progress to realize an example with the PALETTE service BayFac (see D.KNO.04 et D.KNO.06) as a knowledge provider.

4.6.1 Architecture

The different parts of the architecture, as illustrated by Figure 40, are presented in this section following the different functions exposed in the section 4.3: “functional specifications.

The first step of the CAKB functioning is the knowledge notification from the services: RSS feeds for the moment. In these notifications, the services provide the list of last actions performed on them, with references to the concerned concepts as shown in the previous section about the RSS. During this step, the CAKB subscribes to the concerned service RSS with the user authentication parameters. The CAKB can ask then to the concerned service its RSS on a regular basis.

In the second step, the CAKB parses the RSS and inserts the knowledge into Generis according to its ontology.

In the third step, the CAKB asks BayFac to classify actions and resources newly inserted. Since an ontology is used, BayFac provides suitable consultation interfaces that are used in the CAKB. The BayFac used is however reduced to its facet search interface. The classification is done in an automatic way since all the annotations have already been specified according to the CAKB ontology.

The CAKB provides two services: the human consultation, detailed in the next section, and the inter-services pathway providing (see “perspectives” section).

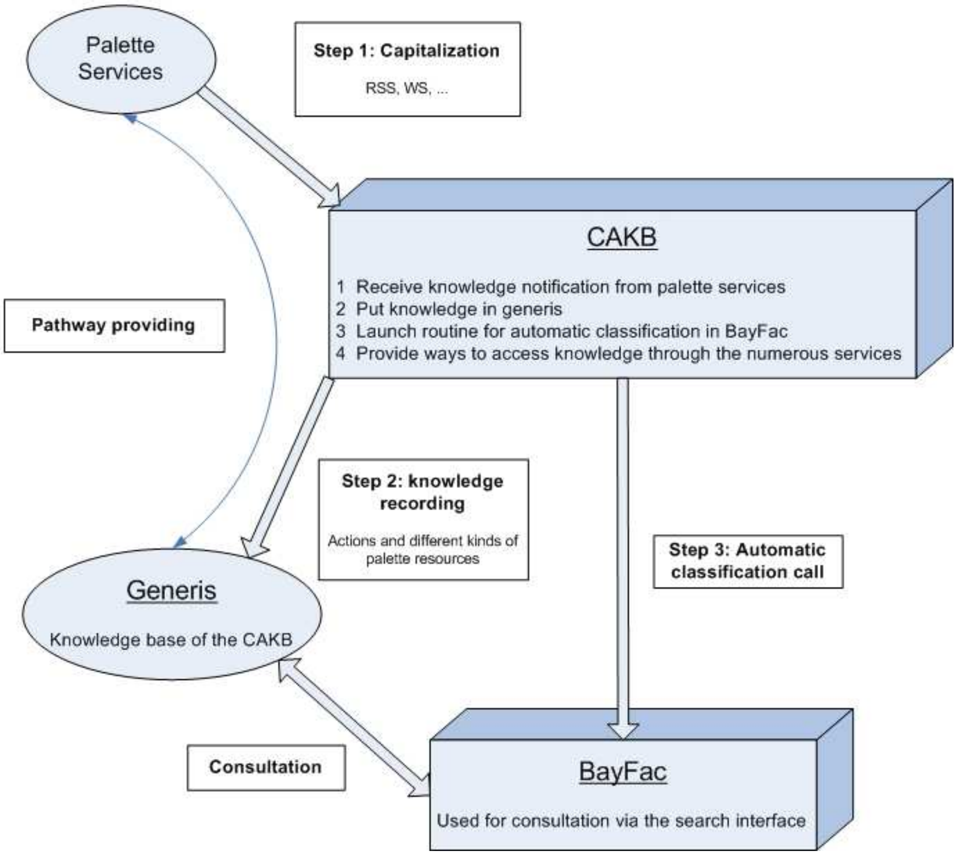
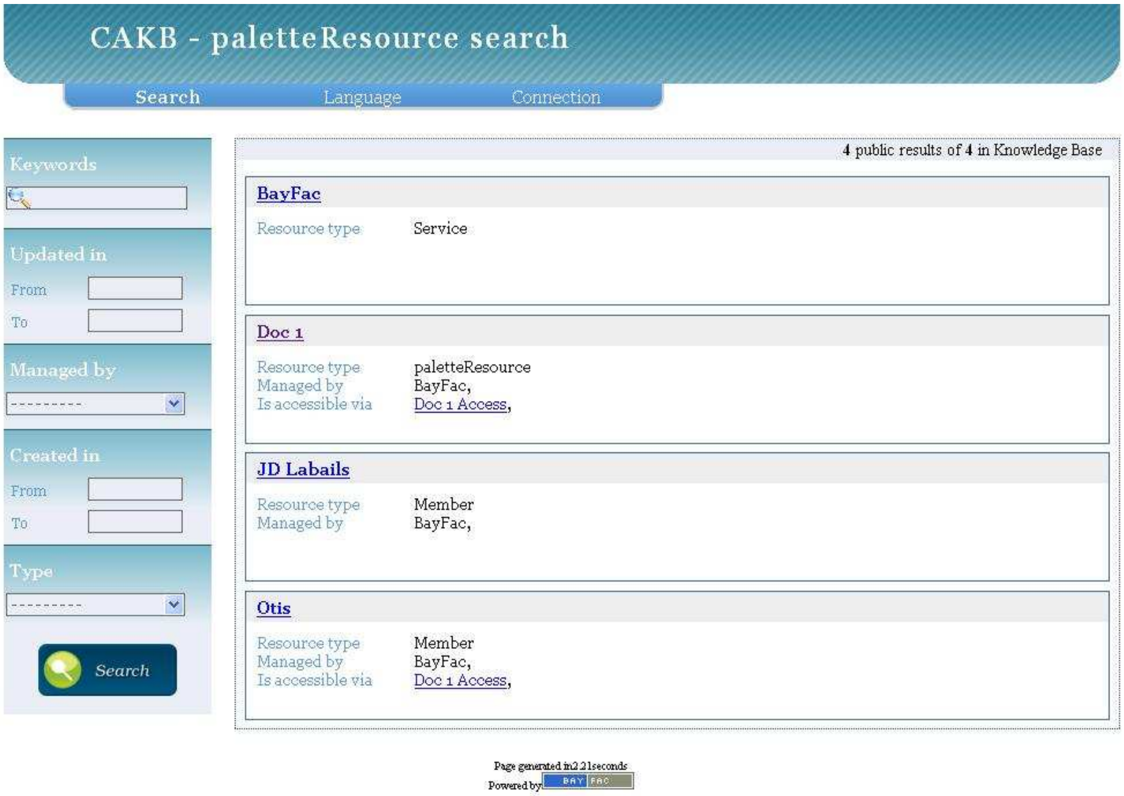

Palette Services
Step 1: Capitalization
RSS, WS, ...
CAKB
1 Receive knowledge notification from palette services
2 Put knowledge in generis
3 Launch routine for automatic classification in BayFac
4 Provide ways to access knowledge through the numerous services
Pathway providing
Step 2: knowledge recording
Actions and different kinds of palette resources
Step 3: Automatic classification call
Generis
Knowledge base of the CAKB
Consultation
BayFac
Used for consultation via the search interface

Figure 40: Architectural view of the different steps

4.6.2 Consultation interfaces

In order to provide to end-users a way to consult the actions and the resources capitalized by the CAKB, a restricted version of BayFac has been used. This enables the consultation of the PALETTE resources currently available in the different PALETTE services of a CoP, as illustrated by Figure 41. For a proof of concept, some facets have been chosen arbitrarily to give examples of the possible criteria. The chosen facets are:

- The service managing the searched resources,
- A period of time in which the search resources have been updated,
- A period of time in which the search resources have been created and
- The type of resource (member, document…)



4 public results of 4 in Knowledge Base

BayFac
Resource type Service

Doc 1
Resource type paletteResource
Managed by BayFac,
Is accessible via Doc 1 Access,

JD Labails
Resource type Member
Managed by BayFac,

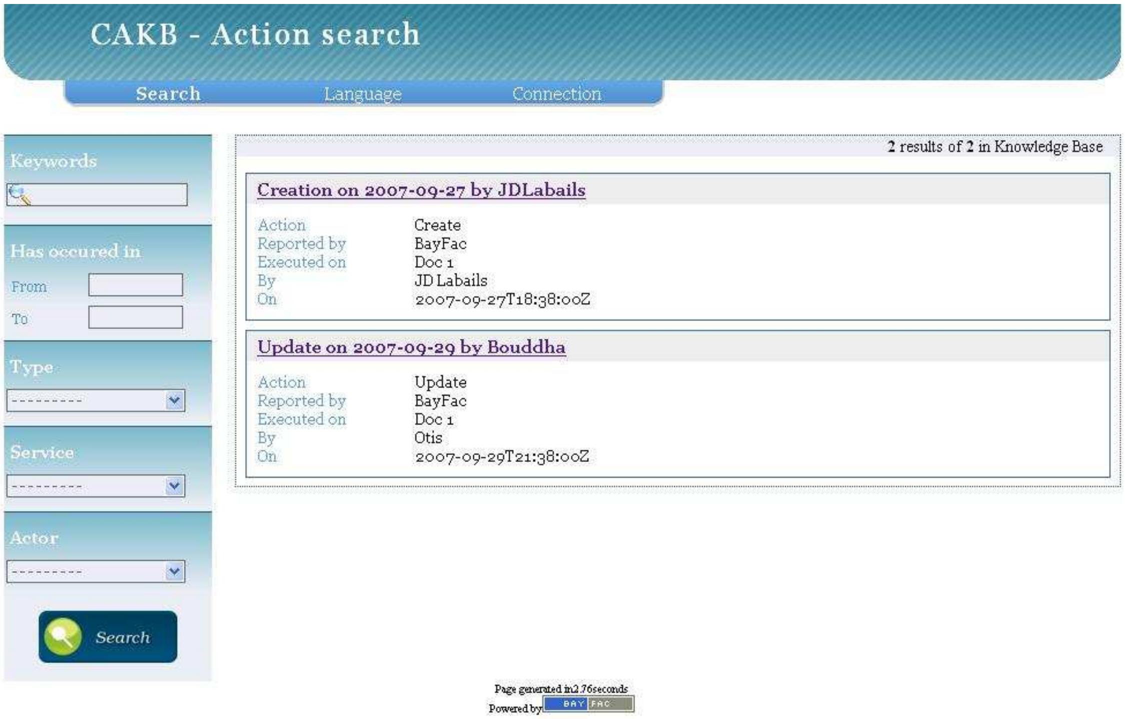
Otis
Resource type Member
Managed by BayFac,
Is accessible via Doc 1 Access,

Page generated in2.21seconds
Powered by BAY FAC

Figure 41: PALETTE resource search view of the BayFac plugged in the CAKB ontology

It also allows consulting the actions on resources. The lifetime of these actions has to be defined since the number of “Action” instances could quickly be very large (one for each read document for example). For this second interface illustrated in Figure 42, the chosen facets are:

- The type of actions (Create, Read, Update or Delete),
- The service on which the searched actions have been performed,
- The member who has performed them, and
- The period of time during which the searched actions have been performed.



2 results of 2 in Knowledge Base

Creation on 2007-09-27 by JDLabails

Action	Create
Reported by	BayFac
Executed on	Doc 1
By	JD Labails
On	2007-09-27T18:38:00Z

Update on 2007-09-29 by Bouddha

Action	Update
Reported by	BayFac
Executed on	Doc 1
By	Otis
On	2007-09-29T21:38:00Z

Page generated in2.76seconds
Powered by BAY FAC

Figure 42: Action search view of the BayFac plugged in the CAKB ontology

5 Perspectives and Future Work

This section briefly discusses the remaining issues and perspectives for each of the three tools of the PSDF presented in this report.

5.1 Future of the Service Browser

The consistency of the first version of the PSB can be increased. In particular, tags and category-based indexing and search should be developed on the PSRF to be made available on the PSB. As for the PSP, a single sign-on module would also be needed. Its non-presence has lead to the necessary storage of several logins and passwords in a file, though this could have been avoided.

Usability still needs to be tested and evaluated based on users feedback. Except for that point and some technical improvements like described in the former paragraph, no major modifications are planned.

From an end-user point of view, some improvements could be made at the service description level. Though this does not concerns directly the PSB as it simply reflects the information provided in the services descriptions, it is important to notice that those descriptions have been made by services providers and could take benefit from behing enhanced with some end-users feedbacks. An idea would be to add little use-cases illustrating how a given service can be used. Feedbacks from CoPs’ members reported in deliverable D.PAR.04, on each of the PALETTE services can be a starting point to include such use cases in the services description. The extension of this in the PSB would be to allow users to add themselves such feedback so that other users could benefit from it. In the case of a CoP where each member can choose himself the tools that she/he wants to use, this collaborative approach could greatly help.

Finally, the PSB could also be extended to offer more functions for the users, which are also service providers or potential widget developers. For instance they could use it to learn more about the available services programming interfaces or to update the service descriptions they have registered in the PSRF.

5.2 Open issues concerning the PALETTE Service Portal

Additionally to the work concerning the portal, we are also exploring the browser extension method with the LinkWidget extension for integrating knowledge management services with other services. It will be described in another deliverable in WP3. Some experiments have also been done with bookmarklets, because we want to keep that option opened during the final step of scenario elaboration and implementation, as it is a lightweight approach ideal for rapid prototyping.

Wiget display performance, even though has been evaluated as fast and responsive, suffers from a known browser bug, documented as Bug 13574 in our internal Webkit Bugzilla (URL: IFRB). The problem relies in the fact that the content of an iframe is reloaded each time the iframe is moved in the DOM. Since all the widgets in the portal are contained within iframes, each time a widget is moved through the Drag and Drop operation, its content is refreshed. Since a widget is in a transitional state during the operation, which corresponds to a specific location in the DOM, the content of the widgets is even refreshed two times. This is less of a problem if the content of the widget is only XHTML and JavaScript, but it may result in serious performance flaws on slower machines if the content is a heavy Flash file or a Java applet. Another side-effect is that the content of a widget, such as user entries in a form, are lost when a widget is moved within the interface, since the widget is reloaded and re-initialised in its starting state.

Another issue is the weak control over the content of the local widget archive. Even though for security reasons, server-side scripting is disabled for local widgets, the archive might contain other malicious files, which are uploaded on the server without verification. It might be worthwhile to implement stronger security measures for the widget upload procedure, such as automatic anti-virus detection and restricting the content of the archive to only a few different file types.

5.3 Perspectives for the CAKB

The Cross-Awareness Knowledge Base aims at providing a means to enhance interoperability and cross-service awareness. A proof of concept (see annexe 8.9) illustrates it. The following perspectives are considered for future developments.

A cross-service search has been developed in the frame of D.IMP.05. It could be inserted into the CAKB consultation user interface instead of the traditional keyword search. That will be more relevant since the current keyword search is mostly based on the documents' content. In the case of the CAKB, the content is not exploited since only metadata are stored. The resulting single search service, planned as a generic scenario for IP3, would combine the keyword search on data of each service via their web services, and the CAKB for search on metadata.

The communication protocol between the CAKB and the services is not yet fully specified, but a big part is achieved. The pathway is only accessible, for the moment, through the "isAccessibleVia" property having as range a literal being the URI of the resource, or the code to call the resource provider web service with the appropriate parameters. This should be enhanced for efficient inter-service communication.

A storage space (as proposed in IP3, single store scenario), can be made available for services willing to share their resources on a central repository to avoid duplication and possible problems of synchronization. This storage space could be linked to the CAKB, and notify it of its state changes (actions performed by users on files).

The authentication of the CAKB to the different services remains an issue. The CAKB is not responsible for a central registration/authentication service for tools and users, but PALETTE CoPs and PALETTE services need it. To bypass the problem, we propose a subscription system for CAKB-users as a possible solution. When a user (as well as a service) subscribes his CAKB to a service to get knowledge (RSS feeds), he provides his authentication parameters for this service (this user must already have an account on this service). When he consults his CAKB, this latest is "aware", according to the user rights, of what happened on each resource of each subscribed service since it has taken the identity of its user (see section 4.5.3). To deploy this solution, the architecture of the CAKB and its consultation interfaces have to be enhanced to provide a CAKB per user or group of user. When a system like SSO will be available on PALETTE, the CAKB will respect it.

Finally, we plan some improvements at the UI and at the ontology levels. In next releases, the UI could incorporate a timeline, as the Simile one (URL: TMS). The timeline would be inserted in the description of each document. The four types of actions would be differentiated by their color (border or title area) to augment the legibility of the consultation of actions. Concerning the CAKB ontology, one example of enhancement is the definition of a concept allowing to know which service manages a particular annotation. This concept could be a subclass of "rdf:property" or just a property having "rdf:property" as domain.

6 Conclusion

The three tools presented here are the first version of the PALETTE Services Delivery Framework. CoP members can now access to services provided by PALETTE through the PSB. With the PSP, they are able to have their own personalized web portal giving a centralized acces to the PALETTE services they use. Finally, CoP members also have the possibility to be aware of actions performed on the knowledge shared among tools they use thanks to the CAKB. These tools answer directly to some important concerns of CoPs, which we listed in the introduction of this document. Their suitability can also be demonstrated by taking some concepts or techniques that have been reported in D.PAR.04 as being possible ways to answer main CoPs' concerns. Among them, we can cite the use of RSS for event notification and awareness, which the CAKB uses to aggregate information provided by registered PALETTE services. Another example would be the similitudes with Facebook, heavily cited in D.PAR.04: like it, the PSP provides a way to construct a personal workspace (web portal) using mini applications, and awareness in particular can be provided through a widget by the CAKB.

From a technical point of view, a main concern, also identified by CoPs as reported in D.PAR.04 (p74) is the problem of the multiple sign-on, said in D.PAR.04 as being "a limiting factor for the use of services available on the Web today". The PSDF together with the palette of services will manage this issue by implementing a single sign-on mechanism, allowing to sign once for all and having access to any service (depending on the actual access rights defined).

The next steps will be now for service providers to register all their PALETTE and web services in the PSRF, so that they can be accessed from the PSB, and develop a series of widgets that will enrich the portal and actually make it a ready-to-use tool providing to CoPs a single access point to the entire PALETTE of services offered by the project. The same thing is true for the CAKB: PALETTE service providers have now to provide RSS feeds so that the CAKB fills itself with useful knowledge.

The main developments to come for the PSDF will concern interoperation and composition of PALETTE services at the presentation level. Some alternatives to the use of the PSP are currently being studied, namely pair-wise integration of two services, but the main research will concern the handling of event-based interactions between widgets, so that action chaining can be performed in relation with the generic scenarios to be developed for IP3.

7 References

7.1 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/

Daniel, F., Matera, M., Yu, J., Benatallah, B., Saint-Paul, R. and Casati, F. (2007). Understanding UI Integration. A survey of problems, technologies, and opportunities. In IEEE Internet Computing, May-June 2007, p. 59-66.

Duerst and Suignard, Eds. (2005). RFC3987. Internationalized Resource Identifiers (IRIs), IETF, January 2005.

Van Kesteren A. (2008). Access Control for Cross-site Requests. W3C Working Draft 14 February 2008, available at www.w3.org/TR/access-control/

Van Kesteren A. and Caceres M., Eds., (2007). Widgets 1.0. W3C Working Draft 13 October 2007, available at www.w3.org/TR/widgets/

Garrett, JJ. (2002). A visual vocabulary for describing information architecture and interaction design. Version 1.1b (6 March 2002), available at www.jjg.net/ia/visvocab/

7.2 Webography

All the Web references have been accessed in March 2008.

ACC W3C Access Control for Cross-site Requests, at http://www.w3.org/TR/access-control/

ATB AddThis social bookmark widget, at http://www.addthis.com/

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

CAKBT Test CAKB, at http://sim.tudor.lu/palette/BayFacCAKB

CGL Cragislist, at http://www.craigslist.org/

CSS http://www.w3.org/TR/CSS21/. Bert Bos, Ian Hickson, Tantek Çelik, Håkon Wium Lie. W3C, April 2006.

DCMI DCMI Namespace for the Dublin Core Metadata Element at http://purl.org/dc/elements/1.1/

FAO Firefox Add-ons: Browse Extensions by Category, at https://addons.mozilla.org/en-US/firefox/browse/type:1

FCB Facebook, at www.facebook.com/

FLD Architectural Styles and the Design of Network-based Software Architectures http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

FLK Flickr, Photo sharing, at www.flickr.com/

GCA Google Chart API, at code.google.com/apis/chart/

GMA Google Maps API, at code.google.com/apis/maps/

GSA Welcome to Google Apps, at www.google.com/a/help/intl/en/index.html

H4S HTML 4 Scripts: http://www.w3.org/TR/html4/interact/scripts.html#h-18.2.3

HMC Housing Maps, at www.housingmaps.com/

HTTPC Pear Http-Client Package: http://pear.php.net/package/HTTP_Client

IFRB Bug 13574: http://bugs.webkit.org/show_bug.cgi?id=13574

JQ jQuery library: http://jquery.com/

JQI jQuery interface website: http://interface.eyecon.ro/

JSON JavaScript Object Notation: http://json.org/

JSR168 JSR 168: http://www.jcp.org/en/jsr/detail?id=168

MDB2 Pear MDB2 Package: http://pear.php.net/package/MDB2

MED Me.dium social software Web surfing, at me.dium.com/

NVB Netvibes, at www.netvibes.com/

OPR Operator Firefox extension, at https://addons.mozilla.org/fr/firefox/addon/4106

OSA OpenSocial Group, at code.google.com/apis/opensocial/

OWS Opera Widget Information File Syntax: http://oxine.opera.com/widgets/documentation/widget-configuration.html

PIC Picnik Photo editing, at www.picnik.com/

PPJ POPJisyo.com Bookmarklets, at www.popjisyo.com/WebHint/en/help/Bookmarklet.aspx

PSN Plaxo. Stay in touch with the people you care about, at www.plaxo.com/

PSB PALETTE Service Browser at http://sim.tudor.lu/palette/srvBrowser/index.php

PSP PALETTE Service Portal at http://sim.tudor.lu/portal/

PSRF PALETTE Service Registry Framework at http://paletteregistry.cti.gr/

RDFS RDF Schema at http://www.w3.org/2000/01/rdf-schema

RDFSS RDF Site Summary (RSS) 1.0 at http://purl.org/rss/1.0/spec

REST REST article on Wikipedia: http://en.wikipedia.org/wiki/REST

TMS Timeline definition at http://simile.mit.edu/timeline/

VWD Viewdle, Facial Recognition, at www.viewdle.com/

W10 W3C Widgets 1.0: http://www.w3.org/TR/widgets/

W10C W3C Widget Configuration File: http://www.w3.org/TR/widgets/#configuration

W10F W3C Widgets 1.0 File Format: http://www.w3.org/TR/widgets/#file-format

W10R Widgets 1.0 Requirements: http://www.w3.org/TR/WAPF-REQ/

WD Desktop Widgets: http://en.wikipedia.org/wiki/Widget_engine#Desktop_widgets

WEBARCH Architecture of the Web W3C Recommendations: http://www.w3.org/TR/2004/REC-webarch-20041215/

WWID Wikipedia - Web Widget: http://en.wikipedia.org/wiki/Web_widget

XHR W3C XMLHttpRequest Object: http://www.w3.org/TR/XMLHttpRequest/

XSDID XML Schema ID: http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/#ID

ZDN ZDNET Download zone at downloads.zdnet.com/

8 Annexes

8.1 PSRF web services API

The following tables present the list of the PSRF Web Methods.

Web Methods: Account Management

Web Method: IdentifyAccount
Description: Identifies an Account by name and password. **Input**: **name**: The name of the account. **password**: The password of the account. **Output**: If the identification succeeded returns 1 else 0. **Exception:** If operation fails, operation error is returned.
Web Method: LoadAccountByID
Description: Loads an Account from Registry by its id. This operation requires the Administrator Credentials for the requestor. **Input**: **id_account**: The id of the account. **Output**: Returns the Account filled up with the Account data. **Exception:** If no data then it returns 0. If operation fails, operation error is returned.
Web Method: LoadAccountByNamePass
Description: Loads an Account from Registry by its Name and Password. **Input**: **name**: The name of the account. **password**: The password of the account. **Output**: Returns the Account filled up with the Account data. **Exception:** If no data then it returns 0. If operation fails, operation error is returned.
Web Method: CreateAccount
Description: Creates an Account in Registry. This operation requires the Administrator Credentials for the requestor. **Input**: An XML Account Object should be created. The requestor, will fill only the required elements.

id_account: Null Value.

name: The name of the account.

password: The password of the account.

email: The email of the account.

affiliation: The affiliation of the account.

type: The type of the account: 1 for Application or Simple User, 2 for Provider, 3 for Administrator

Output: Returns the id of the created Account as the ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: UpdateAccount

Description: Updates an Account in Registry. This operation requires the Administrator Credentials for the requestor.

Input:

An XML Account Object should be created. The requestor will fill only the required elements.

id_account: The id of the account which will be updated.

name: The name of the account.

password: The password of the account.

email: The email of the account.

affiliation: The affiliation of the account.

type: Null Value

Output: Returns 1 if update was successful.

Exception: If operation fails, operation error is returned.

Web Method: DeleteAccount

Description: 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.

Input:

id_account: The id of the account which will be deleted.

Output: Returns 1 if the Account was deleted.

Exception: If operation fails, operation error is returned.

Web Method: CheckEmail

Description: Check for the existence of an email. This operation requires the Administrator Credentials for the requestor.

Input:

email: The email.

Output: Returns 1 if emails exists, otherwise 0.

Exception: If operation fails, operation error is returned.

Web Method: CheckName

Description: Check for the existence of an Account Name. This operation requires the Administrator Credentials for the requestor.

Input:

name: The account name.

Output: Returns 1 if name exists, otherwise 0.

Exception: If operation fails, operation error is returned.

Web Method: LoadAccountList

Description: Loads a List of all active Accounts of the registry. This operation requires the Administrator Credentials for the requestor.

Input: No Params

Output: Returns the Account List filled up with the Account data and 1 as the ResultCode.

Exception: If operation fails, operation error is returned.

Web Methods: Service Management

Web Method: CreateService

Description: Creates a Service Description in Registry. This operation requires the Administrator or the Service Provider Credentials for the requestor.

Input: An XML Account Object should be created. The requestor, will fill only the required elements.

id_service: Null Value.

account_id: The id of the creator's account.

status: The status of the service: 1 for not published, 2 for published.

Xml_content: a base64 encoded string of the XML document with the service description.

Output: Returns the id of the created Service as the ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: UpdateService

Description: 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.

Input: An XML Account Object should be created. The requestor, will fill only the required elements.

id_service: The id of the Service Record.

account_id: Null Value.

status: The status of the service: 1 for not published, 2 for published.

Xml_content: a base64 encoded string of the XML document with the service description.

Output: Returns 0 as the ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: DeleteService

Description: 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.

Input:

id_service: The id of the Service Record.

Output: Returns 0 as the ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: LoadServiceByID

Description: Loads a Service Record from Registry.

Input:

id_service: The id of the Service Record.

Output: Returns a Service Description together with 0 as ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: PublishService

Description: Publishes the Service Record. (Makes it available to public)

Input:

id_service: The id of the Service Record.

Output: Returns 0 as ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: UnpublishService

Description: Unpublishes the Service Record. (Makes it not-available to public)

Input:

id_service: The id of the Service Record.

Output: Returns 0 as ResultCode Number.

Exception: If operation fails, operation error is returned.

Web Method: LoadServiceList

Description: Loads a List of all active Services of the registry.

Input: No Params

Output: Returns a list of Service Records together with 1 as ResultCode if all ok.

Exception: If operation fails, operation error is returned.

Web Method: LoadServiceListByAccount

Description: 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.

Input:

id_account: The id of the account.

Output: Returns a list of Service Records together with 1 as ResultCode if all ok.

Exception: If operation fails, operation error is returned.

Web Method: SearchServices

Description: Searches all published Services of the registry.

Input:

name: The name of the service.

provider: The name of the service provider

category: The category of the service (e.g. Mediation, Collaboration etc.)

type: The type of the service (webService, standAloneApp, webApplication)

contributor: The name of a person of the service development team.

keyword: Keyword(s) from a list of service keywords (multilingual)

description: Word(s) from the service descriptions (multilingual)

orderByPublicationDate: Sort by PublicationDate Desc (1 or 0).

Param elements are mandatory. Params may not be filled. When more than one params have values they are linked with the operand "AND". The string matching is based on a substring case insensitive text search.

Output: Returns a list of Service Records together with 1 as ResultCode if all ok.

Exception: If operation fails, operation error is returned.

Web Method: SearchServicesGlobal

Description: Searches all published Services of the registry using a global search given a keyword.

Input:

key: The keyword.

Output: Returns a list of Service Records together with 1 as ResultCode if all ok.

Exception: If operation fails, operation error is returned.

8.2 PALETTE Widget format specification

8.2.1 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.

8.2.2 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).

8.2.3 Widget files

Every widget *must* define the following two files:

<u>The *config.xml* file:</u>

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.

<u>An index file:</u>

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.

8.2.4 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_location>
```

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>
```

8.2.5 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 o;*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.

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 */
        }
```

8.2.6 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.

8.3 Widget descriptor schema

8.3.1 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>
```

8.3.2 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>
```

8.4 PALETTE Widgets 1.0 Tutorial

8.4.1 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.

8.4.2 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>
```

8.4.3 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 as 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</palette
: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>
```

8.5 Widget formats state of the art

Java Portlets

Java Portlets are pluggable user interface components that are managed and displayed in a web portal. They produce fragments of markup language that are aggregated by a portal. Portlet standards, such as the Java Portlet Specification JSR168 (URL: JSR168) enable software developers to create portlets that can be plugged in any portal that respects the standard.

The Java Community Process introduced the Java Portlet Specification as a convenient programming model for portlet developers. It supports the Model-View-Controller pattern by defining two phases of action processing and rendering, portlet modes that the portal can use to tell the portlet what content it should generate or what task it should perform, window states, portlet data models and a packaging format.

The JSR168 specification represents the same concept than the following widget formats, with the difference that the technology and environment used is Java.

W3C Widgets

The W3C Web Application Formats Working Group develops languages for client-side Web Application development. They published the Public Working Draft of the Widgets 1.0 specification the 13 October 2007 and while their standard is not yet respected by any browser manufacturer, the Widgets 1.0 specification has served as a design basis for a lot of other different specs. While many details are still missing, the advantage of the W3C Widgets 1.0 specification is that it focuses only on the generalities and key principles while leaving some of the more technical aspects aside. As such, the Widgets 1.0 specification serves as a perfect starting point for creating more complicated specifications or adding new, application specific, features. The W3C intend to address all requirements of the Widgets 1.0 Requirements (URL: W10R) document and standardize the way client-side Web applications are to be scripted, digitally signed, secured, packaged and deployed in a device independent way.

While the type of widgets addressed by the W3C Widgets 1.0 spec are downloaded and installed locally on the client machine, it is nevertheless possible to analyse the packaging and structure for our own requirements.

File Format:

Widgets are a bundled archive of files, as specified by the zip file format specification. The details on compression methods and access prevention might vary in the future. The widget files should be at the root of the .zip file with any associated resources in the same directory or subdirectories.

Widget Files:

Every widget must contain the following two files:

1. A manifest config.xml containing information necessary to initialise the widget. This file contains information about the widget's name and geometry and could contain meta information about the widget, such as the author or a human-readable description.
2. The main document for the actual widget, called index.html. This document is displayed in the viewport whose main properties are established by the config.xml file. This HTML document can reference external content in the same way that regular Web pages can.

Widget Configuration File:

The widget configuration manifest config.xml currently allows to specify the widget name and dimensions as well as additional meta information: the author (name, email, url), widget description, widget icon, unique widget id and the security details.

Widget Content File:

The main widget document index.html is an HTML document with much the same constraints as a regular Web page. An HTML document allows including and executing interpretable scripting languages, such as JavaScript. It is not possible to include server-side scripting languages such as PHP.

Widget Scripting Interface:

Through a special *widget* object, the user agent exposes functionalities specific to widgets that should not be available to regular Web pages. Among the object functionalities are methods to access and store preferences that are saved locally for future usage. This allows customizing the widget based on user preferences.

A second *window* object allows resizing and moving the widget by scripting.

Model-View-Controller Design Pattern:

The Widgets 1.0 specification applies the MVC model to their widget format. The accessible data (model) is stored locally through the *widget* object, which is part of the controller. The data presentation and user interaction is handled by the HTML markup (possibly using CSS to separate the layout from the structure). Unfortunately, it is not possible to define the entire widget data (user preferences) through the configuration file but it is necessary to change them through the controller.

Opera Widgets

Opera Widgets are very similar to W3C Widgets, considering that the W3C Widgets 1.0 Working Draft is based on the initial version of the document “Opera Widgets 1.0” written by Arve Bersvendsen and Charles McCathieNevile Similar to W3C Widgets, Opera Widgets are small web applications that run directly on a user's desktop, if the Opera browser is installed, and thus are rendered without the browser chrome.

File Format:

Opera Widgets are packaged in the zip format and have their extension renamed to “.wdgt”. The widget files must be at the root of the archive while other files can be located anywhere within the widget archive file.

Widget Files:

Opera Widgets must contain the following two files:

1. A manifest config.xml which is similar in nature to the W3C Widgets 1.0 specification.
2. The main document for the widget, called index.html.

Widget Configuration File:

Because of the common origin, the Opera Widget configuration file is nearly identical to the W3C Widgets 1.0 config.xml file and documented in the Opera Widget information file syntax (URL: OWS). The allowed XML elements are similar, with only a few changes, such as the id element being mandatory and having three child elements.

Widget Content File:

Additionally to the functionalities offered by HTML and JavaScript, Opera Widgets can be created using content that Opera handles natively, such as SVG or XML files.

Widget Scripting Interface:

Opera Widgets have a *widget* object available through JavaScript that allows to access widget-specific functionality, such as a permanent storage for its settings and downloaded data.

Model-View-Controller Design Pattern:

The design similarities with the W3C Widgets 1.0 specification make the Opera Widgets format compatible with the MVC design pattern, for the very same reasons.

Google Gadgets

Google Gadgets are little applications used on the Google Customized Homepage, called Google IG, and even within the Desktop Application Google Desktop. Google Gadgets are written using the Google Gadgets API.

File Format:

A Google Gadget is a single XML file. External content such as additional CSS files, external script or images have to be loaded from a remote server.

Widget Files:

A Google Gadget is a single XML file that consists of three separate sections:

The Content Section: this section contains the business and presentation logic that determine the functionalities and the appearance of the gadget

The User Preferences: this section defines controls that allow users to specify settings for the gadget.

The Gadget Preferences: this section specifies characteristics of the gadget such as the dimensions and meta information

Widget Scripting Interface:

The Google Gadgets API supports several JavaScript libraries that offer gadget-specific functionalities:

- Core JavaScript library: this library includes access to the user preferences as well as several functionalities commonly offered by Ajax libraries, such as XMLHttpRequest wrapper functions, onload handlers or utility functions.
- Feature-specific JavaScript libraries: the feature-specific libraries include advanced gadget-specific functionalities and several Ajax effects, such as tabs, drag&drop or grid support.

Model-View-Controller Design Pattern:

Google Gadgets make use of the Ajax approach to Rich Internet Applications to create dynamic content. This approach separates the produced content (view) from the content creation, which is usually done in JavaScript (controller). Additionally, the XML file allows to specify the user preferences in a dedicated section, which corresponds to the definition of the gadget model. Thus, Google Gadgets respect the MVC design pattern and by allowing to define the gadget model within the configuration file, makes it even easier to design and create Google Gadgets.

Netvibes Universal Widgets

Netvibes Universal Widgets are created using the Netvibes Univeral Widgets API (UWA), which is the new Netvibes API. They attempt to make Netvibes Widgets available on every possible platform, not only Netvibes but also Google IG, Apple Dashboard and many others. To achieve compatibility between several different widget platforms, an UWA widget includes a special open-source JavaScript runtime that produces platform-specific content depending on the runtime environment. An UWA widget included within the Google IG platform is transformed with respect to the Google Gadgets API, while on the Opera platform, the transformation is Opera-specific.

File Format:

An UWA widget must use only a single, valid and static XHTML file. This file must be XML well formed, UTF-8 encoded and include the Netvibes widget namespace. External content must be loaded from a remote server using JavaScript/Ajax methods.

Widget Files:

An UWA is a single XHTML file that consists of several separate sections:

- Metadata: the widget metadata is stored in the head part of the XHTML file, using the standard meta tags.
- Preferences: preferences are declared using the *preferences* tags, placed in the head part of the XHTML file, in the netvibes widget namespace.
- Content: the content includes the emulation files (a CSS file and a JavaScript file to simulate the Netvibes environment), the inlined scripting and styling sections and the static XHTML body section.

Widget Scripting Interface:

To be as compatible as possible, UWA widgets use a special *widget* object that replaces the document DOM object. The *widget* object includes event handling, methods to modify the content of the widget, extensions of the HTML DOM, widget-specific functionalities such as access to the user preferences and Ajax functionalities.

Model-View Controller Design Pattern

The Netvibes UWA specification has been designed with the MVC pattern as a guideline. Their documentation includes that the model part is the declaration of the preferences, the controller part is the definition of the logic in JavaScript and the view part is the definition of the layout in CSS and the structure in XHTML.

Yahoo! Widgets, Microsoft Gadgets and Apple Dashboard Widgets

Our research has shown that there are no significant differences to the widget specifications already analysed. The only interesting point is that Yahoo! Widgets are defined entirely using the Yahoo!

Widgets XML elements and it is not possible to use HTML. As such, Yahoo! Widgets are the only widgets that are not rendered by the browser but require a specific runtime environment, the Yahoo! Widgets Konfabulator. Additionally, Microsoft Gadgets doesn't specify a view section and the entire widget structure and layout is created through the controller, a JavaScript file that contains all the code for creating the widget.

8.6 RDFS of the CAKB ontology

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

<!DOCTYPE rdf:RDF [
       <!ENTITY rdf           ‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’>
       <!ENTITY rdfs          ‘http://www.w3.org/2000/01/rdf-schema#’>
       <!ENTITY owl           ‘http://www.w3.org/2002/07/owl#’ >
       <!ENTITY dc            ‘http://purl.org/dc/elements/1.1/’ >
       <!ENTITY widget        ‘http://www.tao.lu/datatypes/WidgetDefinitions.rdf#’>
       <!ENTITY cakb          ‘http://www.palette.tudor.lu/Ontologies/cakb.rdfs#’ >
       <!ENTITY generis       ‘http://www.tao.lu/Ontologies/generis.rdf#’>
       <!ENTITY xsd           ‘http://www.w3.org/2001/XMLSchema#’ >
]>

<rdf:RDF xmlns:rdf    ="&rdf;"
       xmlns:rdfs     ="&rdfs;"
       xmlns:owl      ="&owl;"
       xmlns:dc       ="&dc;"
       xmlns:xsd      ="&xsd;"
       xmlns:generis  ="&generis;"
       xmlns:cakb     ="&cakb;"
      xmlns:widget    ="&widget;">



<!--**********Action Concept*****************-->



<rdfs:Class rdf:about="&cakb;Action"
       rdfs:label="Action"
       rdfs:subClassOf="&generis;generis_Ressource"
       rdfs:comment="Meta class of all kind of action."/>

<rdfs:Class rdf:about="&cakb;Create"
       rdfs:label="Create"
       rdfs:subClassOf="&cakb;Action"
       rdfs:comment="Creation of a palette resource."/>

<rdfs:Class rdf:about="&cakb;Read"
       rdfs:label="Read"
       rdfs:subClassOf="&cakb;Action"
       rdfs:comment="Reading of a palette resource."/>

<rdfs:Class rdf:about="&cakb;Update"
       rdfs:label="Update"
       rdfs:subClassOf="&cakb;Action"
       rdfs:comment="Updating of a palette resource."/>

<rdfs:Class rdf:about="&cakb;Delete"
       rdfs:label="Delete"
       rdfs:subClassOf="&cakb;Action"
       rdfs:comment="Deleting of a palette resource."/>

<rdf:Property rdf:about="&cakb;isPerformedBy"
       rdfs:label="is Performed By"
       owl:maxCardinality="1"
       rdfs:comment="Link to the member who had performed this action.">

       <rdfs:domain rdf:resource="&cakb;Action"/>
       <rdfs:range rdf:resource="&cakb;Member"/>
       <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>

<rdf:Property rdf:about="&cakb;isReportedBy"
       rdfs:label="is reported by"
       owl:maxCardinality="1"
       rdfs:comment="Link to the service which had send this action.">

       <rdfs:domain rdf:resource="&cakb;Action"/>
       <rdfs:range rdf:resource="&cakb;Service"/>
       <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>

<rdf:Property rdf:about="&cakb;isExecutedOn"
       rdfs:label="is executed On"
       owl:maxCardinality="1"
       rdfs:comment="Link to the data or/and metadata on which this action was executed on.">

       <rdfs:domain rdf:resource="&cakb;Action"/>
       <rdfs:range rdf:resource="&cakb;Resource"/>
       <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>

<rdf:Property rdf:about="&cakb;occursOn"
       rdfs:label="occurs on"
       owl:maxCardinality="1"
       rdfs:comment="Link to the date on which the action has occured.">
```

```
        <rdfs:domain rdf:resource="&cakb;Action"/>
        <rdfs:range rdf:resource="&dc;date"/>
        <widget:widget rdf:resource="&widget;TextBox"/>
</rdf:Property>


<!--************Resource concept***************-->



<rdfs:Class rdf:about="&cakb;Resource"
        rdfs:label="Resource"
        rdfs:comment="Palette resource, i.e. 'Person', 'Activity', 'Document'...">
        <rdfs:subClassOf rdf:resource="&generis;generis_Ressource"/>
</rdfs:Class>

<rdf:Property rdf:about="&cakb;isAccessibleVia"
        rdfs:label="is accessible via"
        owl:maxCardinality="N"
        rdfs:comment="Link the data or meta data to their path way.">

        <rdfs:domain rdf:resource="&cakb;Resource"/>
        <rdfs:range rdf:resource="&cakb;ServicePathWay"/>
        <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>

<rdf:Property rdf:about="&cakb;isManagedBy"
        rdfs:label="is Managed By"
        owl:maxCardinality="N"
        rdfs:comment="Link the annotation to the service which managed it.">

        <rdfs:domain rdf:resource="&cakb;Resource"/>
        <rdfs:range rdf:resource="&cakb;Service"/>
        <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>

<rdf:Property rdf:about="&cakb;matches"
        rdfs:label="matches"
        owl:maxCardinality="N"
        rdfs:comment="Link the annotation to another (form another service) that matches.">

        <rdfs:domain rdf:resource="&cakb;Resource"/>
        <rdfs:range rdf:resource="&cakb;Resource"/>
        <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>
```

```
<rdf:Property rdf:about="&cakb;isDeleted"
       rdfs:label="is deleted"
       owl:maxCardinality="1"
       rdfs:comment="True if the resource is no more available on the service.">

       <rdfs:domain rdf:resource="&cakb;Resource"/>
       <rdfs:range rdf:resource="&generis;Boolean"/>
       <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>


<!--************Member Concept*****************-->



<rdfs:Class rdf:about="&cakb;Member"
       rdfs:label="Member"
       rdfs:comment="Member of a CoP.">
       <rdfs:subClassOf rdf:resource="&cakb;Resource"/>
</rdfs:Class>


<!--*************Service concept*************-->



<rdfs:Class rdf:about="&cakb;Service"
       rdfs:label="Service"
       rdfs:comment="Palette service.">
       <rdfs:subClassOf rdf:resource="&cakb;Resource"/>
</rdfs:Class>

<rdf:Property rdf:about="&cakb;KnowledgeAccess4CAKB"
       rdfs:label="Knowledge Access for CAKB"
       owl:maxCardinality="N"
       rdfs:comment="Link a service to the path way for the CAKB to get its knowledge.">

       <rdfs:domain rdf:resource="&cakb;Service"/>
       <rdfs:range rdf:resource="&cakb;ServicePathWay"/>
       <widget:widget rdf:resource="&widget;ComboBox"/>
</rdf:Property>


<!--*************ServicePathWay Concepts*********-->



<rdfs:Class rdf:about="&cakb;ServicePathWay"
       rdfs:label="Service Path Way"
```

```
        rdfs:comment="Path way to access a particular resource on a particular palette
service.">
        <rdfs:subClassOf rdf:resource="&generis;generis_Ressource"/>
</rdfs:Class>

<rdfs:Class rdf:about="&cakb;URL"
        rdfs:label="URL"
        rdfs:comment="URL service path way.">
        <rdfs:subClassOf rdf:resource="&cakb;ServicePathWay"/>
</rdfs:Class>

<rdfs:Class rdf:about="&cakb;RSS"
        rdfs:label="RSS"
        rdfs:comment="RSS service path way.">
        <rdfs:subClassOf rdf:resource="&cakb;ServicePathWay"/>
</rdfs:Class>

<rdfs:Class rdf:about="&cakb;REST"
        rdfs:label="REST"
        rdfs:comment="REST service path way.">
        <rdfs:subClassOf rdf:resource="&cakb;ServicePathWay"/>
</rdfs:Class>

<rdfs:Class rdf:about="&cakb;SOAP"
        rdfs:label="SOAP"
        rdfs:comment="SOAP service path way.">
        <rdfs:subClassOf rdf:resource="&cakb;ServicePathWay"/>
</rdfs:Class>

</rdf:RDF>
```

8.7 RDFS Description of the CAKB ontology

Hereafter, the documentation about the rdfs file of this ontology.

NB: the namespace *cakb* stands for *http://www.palette.tudor.lu/Ontologies/cakb.rdfs#*.

8.7.1 Classes

Label/***Id***	**Comment**	**SubClassOf**
Action *cakb:Action*	Meta class of all kind of action.	generis_Ressource
Create *cakb:Create*	Creation of a PALETTE resource.	Action
Read *cakb:Read*	Reading of a PALETTE resource.	Action

Label/*Id*	**Comment**	**SubClassOf**
Update *cakb:Update*	Updating of a PALETTE resource.	Action
Delete *cakb:Delete*	Deleting of a PALETTE resource.	Action
Resource *cakb:Resource*	PALETTE resource, i.e. 'Person', 'Activity', 'Document'...	generis_Ressource
Member *cakb:Member*	Member of a CoP.	Resource
Service *cakb:Service*	PALETTE service.	Resource
Service Path Way *cakb:ServicePathWay*	Path way to access a particular resource on a particular PALETTE service.	generis_Ressource
URL *cakb:URL*	URL service pathway.	ServicePathWay
RSS *cakb:RSS*	RSS service pathway.	ServicePathWay
REST *cakb:REST*	REST service pathway.	ServicePathWay
SOAP *cakb:SOAP*	SOAP service pathway.	ServicePathWay

8.7.2 Properties

Label/*Id*	**Comment**	**MaxCard**	**Domain**	**Range**
is Performed By *cakb:isPerformedBy*	Link to the member who had performed this action.	1	Action	Member
is reported by *cakb:isReportedBy*	Link to the service that has send this action.	N	Action	Service
is executed On *cakb:isExecutedOn*	Link to the data or/and metadata on which this action was executed.	1	Action	Resource
occurs on *cakb:occursOn*	Link to the date on which the action has occurred.	1	Action	Dc:date
is accessible via	Link the data or meta data to	N	Resource	ServicePathWay

Label/***Id***	**Comment**	**MaxCard**	**Domain**	**Range**
cakb:isAccessibleVia	their pathway.			
is Managed By *cakb:isManagedBy*	Link the annotation to the service managing it.	N	Resource	Service
matches *cakb:matches*	Link the annotation to another (form another service) that matches.	N	Resource	Resource
Knowledge Access for CAKB *cakb:KnowledgeAccess4CAKB*	Link a service to the path way for the CAKB to get its knowledge.	N	Service	ServicePathWay
is deleted *Cakb:isDeleted*	True if the resource is no more available on the service.	1	Resource	Boolean

8.8 Exemple cakb RSS

```
<?xml version="1.0" encoding="iso-8859-1"?>

<rdf:RDF
          xmlns                 =     "http://purl.org/rss/1.0/"
          xmlns:rdf             =     "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs            =     "http://www.w3.org/2000/01/rdf-schema#"
          xmlns:owl             =     "http://www.w3.org/2002/07/owl#"
          xmlns:cakb            =     "http://www.tudor.lu/Ontologie/CAKB.rdfs"
          xmlns:ExampleBayFac   =     "http://www.tudor.lu/Ontologies/PaletteInstance.rdf"
          xmlns:dc              =     "http://purl.org/dc/elements/1.1/"
          xmlns:bayfac          =     "http://www.palette.tudor.lu/Ontologies/BayFac.rdfs#"
          xmlns:generis         =     "http://www.tao.lu/Ontologies/generis.rdf#"
          xmlns:facet           =     "http://www.palette.tudor.lu/Ontologies/facet.rdfs#"
          xmlns:ocop            =     "http://www.palette.tudor.lu/Ontologies/ocop.rdfs#"
          xmlns:FHE             =     "http://www.tudor.lu/Ontologie/FormaHetice#"
>

          <channel rdf:about="http://www.formahetice.com/xml/bayfac.rss">
                   <title>BayFac4CAKB RSS</title>
                   <link>http://www.formahetice.com/xml/bayfac.rss</link>
                   <description>Notification of BayFac for CAKB updates on
ExampleBayFac</description>
                   <cakb:paletteService>BayFac</cakb:paletteService>

                   <items>
                             <rdf:Seq>
<rdf:li resource="http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Read12021934840"/>
<rdf:li resource="http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934841"/>
<rdf:li resource="http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Read12021934842"/>
<rdf:li resource="http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934843"/>
<rdf:li resource="http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934844"/>
                             </rdf:Seq>
                   </items>
          </channel>


          <item rdf:about =
"http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Read12021934840">
                   <title>Read on 2008-02-05 07:17:00 by Anonymous</title>
                   <link>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195059704046318800
                   </link>
                   <description>Report of the "Read" action on the "D.KNO.03" resource, on the
2008-02-05 07:17:00 and by the user with the login : Anonymous</description>
```

```
                    <dc:date>2008-02-05 07:17:00</dc:date>
                    <dc:type>Read</dc:type>

                    <ocop:member>
                              <dc:identifier>
          http://www.palette.tudor.lu/formahetice#AnonymousBayFacUser
                              </dc:identifier>
                              <dc:title>Anonymous</dc:title>
                    </ocop:member>

                    <cakb:isExecutedOn>
                              <dc:identifier>
          http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059704046318800
                              </dc:identifier>
                              <dc:title>D.KNO.03</dc:title>
                    </cakb:isExecutedOn>
          </item>


          <item rdf:about =
"http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934841">
                    <title>Update on 2008-02-01 19:51:00 by jd</title>
                    <link>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195060142072701700
                    </link>
                    <description>Report of the "Update" action on the "Hemi Cuba" resource, on
the 2008-02-01 19:51:00 and by the user with the login : jd</description>
                    <dc:date>2008-02-01 19:51:00</dc:date>
                    <dc:type>Update</dc:type>

                    <ocop:member>
                              <dc:identifier>
http://www.palette.tudor.lu/formahetice#jdBayFacUser
                              </dc:identifier>
                              <dc:title>jd</dc:title>
                    </ocop:member>

                    <cakb:isExecutedOn>
                              <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195060142072701700
                              </dc:identifier>
                              <dc:title>Hemi Cuba</dc:title>

                              <cakb:isAccessibleVia>
                                        <dc:type>URL</dc:type>
                                        <dc:source>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195060142072701700
                                        </dc:source>
                              </cakb:isAccessibleVia>

                              <rdfs:comment><![CDATA[Looking good in orange!]]></rdfs:comment>

                              <rdfs:label><![CDATA[Hemi Cuba]]></rdfs:label>

                              <FHE:hasMater>
                                        <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059400089712400
                                        </dc:identifier>
                                        <dc:title>Muscle Car</dc:title>
                              </FHE:hasMater>

                              <FHE:hasAuthor>
                                        <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059439062775300
                                        </dc:identifier>
                                        <dc:title>JD Labails</dc:title>
                              </FHE:hasAuthor>

                              <FHE:hasContentType>
                                        <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059455030859100
                                        </dc:identifier>
                                        <dc:title>Fun</dc:title>
                              </FHE:hasContentType>
```

```
                        <FHE:createdOn><![CDATA[13/11/2007]]></FHE:createdOn>

                        <FHE:hasPath>
                                <![CDATA[http://127.0.0.1/bayfac/uploads/hemi.jpg]]>
                        </FHE:hasPath>

                        <facet:postedOn><![CDATA[14/11/2007]]></facet:postedOn>

                        <bayfac:isPrivate>
                                <dc:identifier>
        http://www.tao.lu/Ontologies/generis.rdf#True
                                </dc:identifier>
                                <dc:title>True</dc:title>
                        </bayfac:isPrivate>
                </cakb:isExecutedOn>
        </item>


        <item rdf:about = 
"http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Read12021934842">
                <title>Read on 2008-02-01 19:51:00 by Anonymous</title>
                <link>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195059704046318800
                </link>
                <description>Report of the "Read" action on the "D.KNO.03" resource, on the 
2008-02-01 19:51:00 and by the user with the login : Anonymous</description>
                <dc:date>2008-02-01 19:51:00</dc:date>
                <dc:type>Read</dc:type>

                <ocop:member>
                        <dc:identifier>
http://www.palette.tudor.lu/formahetice#AnonymousBayFacUser
                        </dc:identifier>
                        <dc:title>Anonymous</dc:title>
                </ocop:member>

                <cakb:isExecutedOn>
                        <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059704046318800
                        </dc:identifier>
                        <dc:title>D.KNO.03</dc:title>
                </cakb:isExecutedOn>
        </item>


        <item rdf:about = 
"http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934843">
                <title>Update on 2008-02-01 19:42:00 by jd</title>
                <link>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195060421006947600
                </link>
                <description>Report of the "Update" action on the "Dodge charger" resource, 
on the 2008-02-01 19:42:00 and by the user with the login : jd</description>
                <dc:date>2008-02-01 19:42:00</dc:date>
                <dc:type>Update</dc:type>

                <ocop:member>
                        <dc:identifier>
http://www.palette.tudor.lu/formahetice#jdBayFacUser
                        </dc:identifier>
                        <dc:title>jd</dc:title>
                </ocop:member>

                <cakb:isExecutedOn>
                        <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195060421006947600
                        </dc:identifier>
                        <dc:title>Dodge charger</dc:title>

                        <cakb:isAccessibleVia>
                                <dc:type>URL</dc:type>
                                <dc:source>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195060421006947600
                                </dc:source>
```

```
                             </cakb:isAccessibleVia>

                             <rdfs:comment><![CDATA[Dodge Charger beauty in
sobriety]]></rdfs:comment>

                             <rdfs:label><![CDATA[Dodge charger]]></rdfs:label>

                             <FHE:hasMater>

          <dc:identifier>http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i119505940008971240
0</dc:identifier>
                                       <dc:title>Muscle Car</dc:title>
                             </FHE:hasMater>

                             <FHE:hasAuthor>
                                       <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059439062775300
                                       </dc:identifier>
                                       <dc:title>JD Labails</dc:title>
                             </FHE:hasAuthor>

                             <FHE:hasContentType>
                                       <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195059455030859100
                                       </dc:identifier>
                                       <dc:title>Fun</dc:title>
                             </FHE:hasContentType>

                             <FHE:createdOn><![CDATA[06/11/2007]]></FHE:createdOn>

                             <FHE:hasPath>
<![CDATA[http://127.0.0.1/bayfac/uploads/syl_83_1100192456_dodge_charger_383ci_se___69_.gif]]>
                             </FHE:hasPath>

                             <facet:postedOn><![CDATA[14/11/2007]]></facet:postedOn>

                             <bayfac:isPrivate>
                                       <dc:identifier>
http://www.tao.lu/Ontologies/generis.rdf#False
                                       </dc:identifier>
                                       <dc:title>False</dc:title>
                             </bayfac:isPrivate>
                   </cakb:isExecutedOn>
          </item>


          <item rdf:about =
"http://www.tudor.lu/Ontologies/PaletteInstance.rdf#Update12021934844">
                   <title>Update on 2008-02-01 19:41:00 by jd</title>
                   <link>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195209358063699800
                   </link>
                   <description>Report of the "Update" action on the "Milky way" resource, on
the 2008-02-01 19:41:00 and by the user with the login : jd</description>
                   <dc:date>2008-02-01 19:41:00</dc:date>
                   <dc:type>Update</dc:type>

                   <ocop:member>
                             <dc:identifier>
http://www.palette.tudor.lu/formahetice#jdBayFacUser
                             </dc:identifier>
                             <dc:title>jd</dc:title>
                   </ocop:member>

                   <cakb:isExecutedOn>
                             <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195209358063699800
                             </dc:identifier>
                             <dc:title>Milky way</dc:title>

                             <cakb:isAccessibleVia>
                                       <dc:type>URL</dc:type>
                                       <dc:source>
http://localhost/BayFacCAKB/index.php/rest/fs/Document/instance/http://www.tudor.lu/Ontologies
/PaletteInstance.rdf#i1195209358063699800
                                       </dc:source>
                             </cakb:isAccessibleVia>

                             <rdfs:comment>
<![CDATA[Pluton, dont la d&eacute;signation officielle est (134340) Pluton, est la
deuxi&egrave;me plus grande plan&egrave;te naine connue du syst&egrave;me solaire et le 10e
plus grand astre connu orbitant le Soleil. Originellement consid&eacute;r&eacute;e comme la
plus petite plan&egrave;te du syst&egrave;me solaire, elle fut r&eacute;trograd&eacute;e au
rang de plan&egrave;te naine par l&#039;Union Astronomique Internationale en ao&ucirc;t 2006.
Elle orbite autour du Soleil &agrave; une distance variant entre 29 et 49 unit&eacute;s
astronomiques et appartient &agrave; la ceinture de Kuiper, il s&#039;agit du premier objet
transneptunien &agrave; avoir &eacute;t&eacute; d&eacute;couvert.]]>
                             </rdfs:comment>

                             <rdfs:label><![CDATA[Milky way]]></rdfs:label>

                             <FHE:hasMater>
                                       <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195058927011069700
                                       </dc:identifier>
                                       <dc:title>Astrophysique</dc:title>

                             </FHE:hasMater>

                             <FHE:hasAuthor>
                                       <dc:identifier>
http://www.tudor.lu/Ontologies/PaletteInstance.rdf#i1195209469000575300
                                       </dc:identifier>
                                       <dc:title>Wikipedia</dc:title>

                             </FHE:hasAuthor>

                             <FHE:hasContentType>
                                       <dc:identifier>
http://www.tudor.lu/Ontologie/FormaHetice#EvaluationGrid
                                       </dc:identifier>
                                       <dc:title>Pédagogique</dc:title>
                             </FHE:hasContentType>

                             <FHE:createdOn><![CDATA[07/11/2007]]></FHE:createdOn>

                             <FHE:hasPath>
<![CDATA[http://fr.wikipedia.org/wiki/%28134340%29_Pluton]]>
                             </FHE:hasPath>

                             <facet:postedOn><![CDATA[16/11/2007]]></facet:postedOn>

                             <bayfac:isPrivate>
                                       <dc:identifier>
http://www.tao.lu/Ontologies/generis.rdf#False
                                       </dc:identifier>
                                       <dc:title>False</dc:title>
                             </bayfac:isPrivate>
                    </cakb:isExecutedOn>
          </item>

</rdf:RDF>
```

8.9 CAKB proof of concept

BayFac “CAKB consultation interfaces” URL: http://sim.tudor.lu/palette/BayFacCAKB

BayFac “knowledge provider” URL: http://sim.tudor.lu/palette/BayFac

BayFac “knowledge provider” RSS: http://sim.tudor.lu/palette/BayFac/index.php/Test/test

9 Table of figures