Palette
Pedagogically sustained Adaptive Learning through the Exploitation of Tacit and Explicit Knowledge

Information Society Technologies

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.04 – Updated version of guidelines for development
Towards a Palette Service Platform Architecture

Due date of deliverable: September 30, 2007
Actual submission date: November 15, 2007

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

Organisation name of lead contractor for this deliverable: EPFL

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

Keyword List: architecture, SOA, data mashup, service integration, service interaction, user interface development guidelines, activity-oriented software, resource-oriented software, REST, SOAP

Responsible Partner: EPFL

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
	30 Jul 2007	Draft	Pierre Geneves
	28 Sep2007	Draft	Yannick Naudet, Nikos Karousos
	3 Oct 2007	Draft	All
	4 Oct 2007	Draft	Stéphane Sire, Nikos Karousos
0.9	15 Oct 2007	Mostly Complete for wp5 feedback	Stéphane Sire, Alain Vagner
1.0	19 Oct 2007	Complete for reviewer's feedback	Stéphane Sire
1.1	6 Nov 2007	Revised for SC feedback	All
1.2	12 Nov 2007	Final	Stéphane Sire, Nikos Karacapilidis

Deliverable manager

- Stéphane Sire, EPFL

List of Contributors

- Nikos Karacapilidis, Nikos Karousos, CTI
- Benjamin Gateau, Yannick Naudet, Alain Vagner, Marie-Laure Watrinet, CRP-HT
- Michael Chiu Man Yu, Pierre Geneves, Stéphane Sire, EPFL

List of Evaluators

- Cécile Roisin, INRIA-Grenoble
- Olivier Corby, INRIA-Sophia

Summary

This document updates the guidelines for development of Palette services. Towards this purpose, it continues the analysis of CoP's needs that was initiated in a previous deliverable (D.IMP.03) towards the identification of interoperability requirements and solutions. This is done through an analysis of the same scenarios as the ones which have been synthesized in D.PAR.03. From the identification of the interoperability needs of CoPs, this document then examines some architectures that could support the related interactions between heterogeneous services. Then, it proposes a Palette Service Platform Architecture that properly supports the registration, discovery and delivery of different services. It also makes statements about the types of communication protocols to be used for the interaction between services. The proposed architecture is flexible enough so that it can benefit from different composition techniques in the future to offer composite services (the participatory design methodology is expected to bring up additional needs). Finally, this document makes clear statements about the methods and techniques that should be applied to harmonize the Palette services for the better usability of services when they are combined by an end-user.

1 Introduction

The present updated version of guidelines for development is focused on the integration of Palette services from the point of view of interoperability between several services. The integration is treated at both the software architectural and the user interface (UI) levels. The technical propositions in this document are grounded on an analysis of the current scenarios of interaction of CoPs (as per D.PAR.03).

The main contribution of this document is that it presents a realistic proposition of a global Architecture of the Palette platform in the context of Work Package 5 (WP5): "Implementation of PALETTE Services and Scenarios". The goal is to define a commonly acceptable architecture from all Palette partners as a basis for services declaration, identification and interoperation. The proposition must be a flexible solution, as according to the participatory design methodology, the needs may evolve as the end users are exposed to the technology. In particular, this document focuses on a module-based schema that contains several separated subsystems. The document describes them and analyzes the way of communication between them. The design rationale that led to this proposition is also presented, together with some first technical details concerning the communication protocols, service descriptions and service access procedures. The existence of the proposed architecture assures that the Palette project will support:

- the coexistence of different services;
- the registration of all provided services, and
- the adoption of a common mechanism for services delivery.

This document is targeted at the developers of Palette services to whom it will provide a reference architecture and some necessary steps in order to integrate their services in a composite environment. It also defines some modules that will have to be developed in WP5 to facilitate such an integration. To some extent, this document is also targeted at all the non-technological partners, such as the mediators, as it is built from the scenarios that were elaborated during the participatory design process.

In the following part of the introduction, we give some necessary definitions for the concepts used and a list of useful acronyms. Then, the second section presents an analysis of current interoperability needs as they have emerged in the current version of the scenarios. The third section discusses several architectural styles that could be applied to satisfy these needs and their possible evolutions. The fourth section describes the architecture currently under development. Finally, the fifth section gives some guidelines for homogeneous UI. This last chapter launches a reflection on the different ways to raise the level of UI consistency between the different services during the integration phase. The conclusion summarizes the contribution and sets up an agenda for the continuation of work.

1.1 Definitions

- **Web service. "***The W3C defines a Web service (many sources also capitalize the second word, as in Web Services) as a software system designed to support interoperable Machine to Machine interaction over a network. The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server, a description in the Web Services Description Language (WSDL)."* [http://en.wikipedia.org/wiki/Web_service]
- **PALETTE service.** As defined in D.IMP.03, two kinds of services are considered by the project, namely learning and technological ones. In this document we consider only the latter. With respect to the D.IMP.03 definition, PALETTE technological services are *services* offered to CoPs, *materialized* by one or a composition of functions of software tools provided by PALETTE partners. Technical services that are offered are of different kinds:
  - *Web Applications* (e.g. CoPe_it!, e-Logbook), defined as software applications accessible from the Web.

  - *Web Services*, which are mainly externalized functions of PALETTE services that are Web applications. RESTful Web services will be mainly provided, but SOAP Web services might also.
  - *Composite Web Services*, which are complex Web services defined as a composition of other Web services (e.g. Services integrating functionalities of e-Logbook and CoPe_it!).
  - *Non-Web applications*, which are accessed directly. These ones can potentially access the Web or being dedicated for the Web (e.g. Amaya), but are not deployed on a Web server, contrary to Web applications. Some of them may also be OS specific (e.g. Win32 applications).
- **External service.** Other services that have been developed outside the context of the Palette project but are considered useful for internal usage are defined as external services. Usually, that kind of services are available through common standards such as Web services, however we can meet specific purpose services that are using other protocols such as streaming protocols, online communication protocols, etc.
- **Service Registry.** A registry containing metadata about services but not the services themselves. Those metadata are description of services, containing information dedicated for both humans and/or machines. A service registry is not a repository of services as it does not contain the services themselves, but this is a repository of services descriptions.
- **Data Repository.** A repository for particular data (that can be e.g. services, knowledge, or other data), stores the actual data themselves. In other words, a Repository contains things (data), contrary to a Registry, which contains information about these things (metadata).
- **Single Sign-On.** A methodology that allows users to make only one sign-on while they enter a system allowing them to participate in many tools, services and applications within the system context (without the need to sign-on for each individual tool). It is considered a critical issue in any Service Oriented Environment.
- **Service composition.** Composition is related to: services calls sequencing, collaboration between services, composite services. Orchestration and Choreography can be viewed as compositions.
- **Service orchestration.** "Orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organizations to define a long-lived, transactional, multistep process model. Orchestration always represents control from one party's perspective." (Peltz, 2003). While not being limited to the realization of business processes, service orchestration has the following characteristics:
  - Conception and execution of the composition
  - Defined service execution order.
  - Management of transactions and long running processes.
- **Service choreography.** "Choreography tracks the message sequences among multiple parties and sources – typically the public message exchanges that occur between Web services – rather than a specific business process that a single party executes." (Peltz, 2003). Choreography manages the exchanges between services of different peers (partners).

1.2 Acronyms

- API : Application Programming Interface
- BPEL : Business Process Execution Language
- CAKB : Cross Awareness Knowledge Base
- DOM : Document Object Model

- CoP : Community of Practice
- GUI : Graphical User Interface
- HTTP : HyperText Transfer Protocol
- HMI : Human-Machine Interface
- KM : Knowledge Management
- LDAP : Lightweight Directory Access Protocol
- MMI : Machine-Machine Interface
- N3 : Notation 3 Language for RDF
- N-Triple : A line-based, plain text format for encoding an RDF graph. It was designed to be a fixed subset of N3
- OWL : Web Ontology Language
- PSPA : Palette Service Platform Architecture
- PSRF : Palette Service Registry Framework
- RDF : Resource Description Framework
- RDFS : RDF Schema
- REST : Representational State Transfer
- RIA : Rich Internet Application
- RSS : Really Simple Syndication
- SB : Service Bus
- SMTP : Simple Mail Transfer Protocol
- SOA : Service Oriented Architecture
- UDDI : Universal Description Discovery and Integration
- UI : User Interface
- URI : Universal Resource Identifier
- WADL : Web Application Description Language
- WS-CDL : Web Services Choreography Description Language
- WSDL : Web Service Definition Language
- XHTML : Extensible HyperText Markup Language
- XML : Extensible Markup Language
- XML-RPC : Remote Procedure Call protocol which uses XML to encode its calls and HTTP as a transport mechanism.
- XSLT : (EXtensible Stylesheet Language) XSL Transformations
- XUL : XML User Interface Language

1.3 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 Interoperability needs in Palette

2.1 Analysis of CoPs' scenarios in a service interaction perspective

The participatory design methodology has led to the definition of scenarios, which have been validated by CoPs members. Based on the current version of these scenarios (reported in D.PAR.03), we have identified some possible interactions between services. These interactions are presented below for each CoP. The presentation also summarizes the integration questions that have been raised by the participants and that are reported in the appendix 4 of D.PAR.03. For all scenarios, the version refers to the file stored on the BSCW.

ePrep:

- *Scenario version*: Scenario for the ePrep CoP (version 4).zip (2007-06-14)
- *Palette Services used are*: e-Logbook, SweetWiki, Amaya, LimSee3
- *Interactions:*
    - e-Logbook ← SweetWiki: SweetWiki files are uploaded on e-Logbook as assets.
    - e-Logbook ← Amaya: Amaya files are uploaded on e-Logbook as assets.
    - e-Logbook ← LimSee3: LimSee3 files are uploaded on e-Logbook as assets.
    - e-Logbook → assets: Tags can be added on assets, thus on LimSee3, Amaya uploaded files. Links can be created between assets (see Linking assets and activities section in e-Logbook scenario section, in the ePrep scenario), if these links can have a meaning for other services, they might be shared, as well as tags.

Here, e-Logbook is used as the main platform. Integration questions concern: access to other documents from e-Logbook, information retrieval, tagging with Amaya and SweetWiki (where to store tags?).

Issues for the CoP: the created pedagogical content is distributed in three different stores managed respectively by Amaya, LimSee3 and SweetWiki. The CoP would like to have a tool allowing the retrieval of any content, whatever its location. A cross awareness KB storing links to each file could be a solution.

@pretic:

- *Scenario version*: Unified scenario for @pretic - Using SweetWiki and Amaya.htm (2007-06-29)
- *Palette Services used are*: Amaya, SweetWiki+Corese, (+ECCO)

An ontology has been created for the CoP. This ontology is used to annotate (with a Corese function) SweetWiki pages; it is accessible and editable from SweetWiki.

- *Interactions:*
    - Forum → SweetWiki: SweetWiki is used as main tool, but the information production chain contains a non-Palette service (a forum). Information from the forum is pasted to SweetWiki in a manual way.
    - Amaya - SweetWiki: Exchange of documents (XHTML) is made manually. Enhancement is not foreseen (see D.IMP.03), however, a service of SweetWiki allowing a new page to be created directly from a given XHTML file would allow direct transfer of files from Amaya into SweetWiki. The inverse would also be interesting: i.e. opening an XHTML file with a local editor (e.g. Amaya) from SweetWiki.

Form@HETICE:

- *Scenario version*: June 2007 validated scenario for Form@HETICE.html (2007-07-03)

- *Palette Services used are*: BayFac, Amaya
- *Palette Services foreseen from integration questions*: Limsee3, SweetWiki
- *Interactions:*
    - Amaya - BayFac: No contact specified in the scenario. However, documents created with Amaya can be classified by BayFac.
    - Amaya - SweetWiki: see @Pretic;

Learn-Nett:

- *Scenario version*: scenario_learn_nett_v4.zip (2007-07-04)
- *Palette Services used are*: SweetWiki, CoPe_it!, e-Logbook, LinkWidget, ECCO
- *Non-Palette tools*: Galanet, Moodle, Centra

The CoP uses two global platforms for their needs: Galanet and Moodle, plus a video conferencing tool (Centra). Interactions with Palette services are foreseen through these platforms. A single entry point to used tools / services with single sign-on is explicitely asked.

- *Interactions:*
    - CoPe_it! ← e-Logbook: "Context Aware View" awareness function of e-Logbook invoked from CoPe_it!
    - LinkWidget → cop platforms, SweetWiki: annotations, search (through Corese) on external (e.g. from Moodle) resources or those handled in Palette Services
    - LinkWidget, SweetWiki ← ECCO: Use of ECCO's ontologies (URL: access).
    - SweetWiki, learning platforms (Moodle, Galanet) ← LinkWidget: search interface (LinkWidget is a FireFox plugin) used to search metadata.
    - SweetWiki ← learning platforms: Written conversations are pasted to SweetWiki in a manual way.
    - SweetWiki ← CoPe_it!: Summary of discussion in CoPe_it! are put manually in SweetWiki.
    - SweetWiki ← CoPe_it! / e-Logbook: awareness metadata manually added to summary (see previous point) put in SweetWiki.
- Foreseen:
    - CoPe_it! - e-Logbook: synchronization needed for context awareness view
    - CoPe_it! - KM services: tagging, ontology access and management

Did@cTIC:

- *Scenario version*: Did@ctic_Scenario_v3.html (2007-06-28)
- *Palette Services used are*: Amaya, KM-Widget, DocReuse (Restructuring Service)
- *Palette Services foreseen from integration questions*: LinkWidget
- *Non-Palette tools*: Moodle, Google Calendar

The scenario is based on the wide use of tagging for document classification and retrieval. The tagging tool appears to be KM-Widget, and use of ontology is foreseen (with ECCO).

- *Interactions:*
    - Tagging performed on documents (Amaya's or else), by KM-Widget
- Foreseen:

- DocReuse ← LinkWidget: Invocation of LinkWidget from DocReuse (to allow tagging at restructuration time)
- DocReuse ← learning platforms: Documents from various CoP repositories (including legacy repositories from the last 10 years) are imported to DocReuse for being recomposed

Adira:

For Adira we also found traces of discussion on the BSCW in the file Discussion for Adira.doc

- *Scenario version*: Scenario-for-Adira-V2-06-07 (2007-06-25)
- *Palette Services foreseen*: SweetWiki, CoPe_it!, e-Logbook
- *Non-Palette tools*: mainly desktop applications (Photoshop, Word, Excel, etc.) and Web portal managed by a professional with a specialized client update tool

Adira is a complex (having a complex IT infrastructure) and mature CoP. Needs in term of Palette Services have only been suggested so far, thus the scenario in its current state does not allow identification of composition needs. A particular whish of the CoP is to have a single access to all used services: a Web portal accessible from the CoP Web site.

2.2 Synthesis of CoPs scenarios in terms of service by service possible interactions

Services interactions had been divided into three levels in D.IMP.03:

- transmission of data and metadata between two services at the *information exchange level*;
- direct call of a service function from another service at the *integration level*;
- composition of several services functions at the *composition level*.

The first level dominates in the current version of the scenarios. Only two cases of direct call of a service function from another service are cited. The first one is in the Learn-Nett scenario: the "Context Aware View" function of e-Logbook called from CoPe_it! It also highlights a specific issue, *synchronization* of views, which is a general concern for each information access that needs to be always and automatically up-to-date. The second case of a direct service invocation is in the Did@cTIC scenario: the restructuration of a document by DocReuse called from other services that contain their own repository of documents. No clear case of composition of several services is cited in the current version of scenarios.

It is interesting to notice that the kinds of interactions cited in scenarios concern four areas, in which we recognize the main concerns expressed by CoPs as cited in D.IMP.03:

- Data sharing: a service needs to access to a document, file, or more generally data, created by another service. Different kinds of access / sharing methods are cited: File upload or link, Manual cut/paste of data.
- Metadata sharing: services need to access or manipulate metadata created by other services. Cases cited concern metadata addition, access, sharing, transmission, and management. Examples are: a service adding metadata on a data potentially created by another, a service needing statistics added by another service.
- Knowledge retrieval: services need to retrieve information (data or metadata) created or handled by other services.
- Awareness: this specific area concerns the sharing of specific metadata dedicated to awareness between tools. In the first example of e-Logbook - CoPe_it! awareness is destined to the user, but this can be generalized to awareness for services for example to lock data access when it is being modified.

Altogether, crossing the interactions analyzed CoP by CoP in the previous section, with the interaction areas previously defined in D.IMP.03, we can synthesize, in the tables below, a preliminary (non

exhaustive) list of interactions for each provided Palette Service. When implemented, these interactions will fulfil the needs of scenarios and augment the work of CoPs.

The last column of each table is a suggestion of corresponding service names, in the form of functions to be provided as REST services for instance, so as to contribute to global interoperability. The list of proposed functions will need to be extended for enhanced versions of scenarios or to facilitate the demonstration of some compositions, if further composition needs will be identified later in the project.

Table 1. List of interactions for **e-Logbook** service

Interaction Area	Need or Suggestion	Proposed function
Data sharing	Needs uploading or linking of files from SweetWiki, Amaya, LimSee3	loadDocAsAsset(doc_uri / doc_content): called from SweetWiki, Amaya, LimSee3
Metadata sharing	Could share its assets tags and other metadata with other services (Interest: ePrep)	getMetadataForAsset(asset_uri), getMetadataX()
Metadata sharing	Could need retrieving tags and other metadata put by Amaya and SweetWiki	
Awareness Metadata sharing	Could share awareness statistics (number of member access to assets, tags, evaluations) (Interest: ePrep)	
Awareness	Could share the new member notification function: this can be useful for a cross awareness tool (Interest: ePrep)	
	Needs to call CoPe_it! functions: the choice of the needed functions still needs to be done (Interest: Learn-Nett, Adira)	

Table 2. List of interactions for **SweetWiki** service

Interaction Area	Need or Suggestion	Proposed function
Data sharing	Needs to exchange documents with Amaya	loadDocInPage(doc_uri): called from Amaya to create a page with a document in edition. SweetWiki will need to identify if the document is already in a page or if a new one should be created. The call of this function might make SweetWiki appear in a browser if not already open (the calling service should handle this). This function can be called by Amaya, CoPe_it!
		getPage(page_uri): for Amaya. But this may be too complicated to implement regarding the ease of copy/paste...
Metadata sharing	Needs an access to ECCO ontologies	Availability of ontologies is already ensured, as they are published on the Web and accessible through a URI.

Metadata sharing	Could share its tags and a pointer to the place in the document they refer to, as well as the list of available tags or categories (Interest: ePrep)	getMetadataForDoc(doc_uri), getTagList()
Metadata sharing	Could share the folksonomy created with the tags of a document (Interest: ePrep)	getFolksonomy()
Metadata sharing K retrieval	Should make its tags available for document retrieval from search engines (e.g. Corese) (Interest: Learn-Nett, Did@ctic, ePrep, Adira)	
K Retrieval	Could share its search function: e.g. for searching docs from another tool (e.g. e-Logbook), by tags and keywords (Interest: ePrep)	
Awareness	The RSS field can be exploited for awareness by external tools or e.g. a cross awareness tool (Interest: ePrep)	getAwarenessInformation(asset_uri): to provide awareness metadata concerning an asset or the whole platform. "Context Aware View" used by CoPe_it! ?
Awareness	Could share notifications (notifications of page access, modification, statistics): they are sent by eMail, but this could be useful for global awareness, or by other tools accessing SweetWiki pages. (Interest: @pretic)	see above

Table 3. List of interactions for **Amaya** service

Interaction Area	Need or Suggestion	Proposed function
Data sharing	Needs to exchange documents with SweetWiki	Can be performed with a call to SweetWiki:loadDocInPage()
Metadata sharing	Could share documents' metadata and other annotations (Interest: ePrep). e.g. for Form@HETICE, BayFac could read these metadata (e.g. Author, Creation date, etc.) to pre-classify a document.	Can not be done with a Web service, as Amaya is a standalone application. If metadata are put in the file or referenced in the file by URI, this can be used by other services, if they know the semantics of metadata added by Amaya.
Awareness	Could share modification/ creation notifications: activity information can be useful to a cross awareness tool (Interest: ePrep, Form@HETICE). Amaya sends new document notification by mail; this information could be used for awareness or event-based composition. In the case of Form@HETICE, BayFac could automatically handle the new created document.	

Table 4. List of interactions for **Bayfac** service

Interaction Area	Need	Suggested service
Metadata sharing	Could share classification metadata (Interest: Form@HETICE)	getMetadataForDoc(doc_uri)

K retrieval	Could share its search function (according to facets or not). (Interest: Form@HETICE)	search(facet_values / key_words): For searching documents classified by BayFac from an external tool.

Table 5. List of interactions for **Cope_It!** service

Interaction Area	Need or Suggestion	Proposed function
Awareness	Needs to use the "Context Aware View" function of e-Logbook	
Data sharing	Needs a way to copy the summary of a discussion into SweetWiki	
	Functions are needed by e-Logbook. Still to be identified.	

Table 6. List of interactions for **LinkWidget** service

Interaction Area	Need or Suggestion	Proposed function
Metadata sharing - K retrieval	Needs to be accessible for annotating and searching on resources handled by SweetWiki and Learn-Nett platforms	As a plugin for a Web browser, Firefox, it has been made especially for annotating and searching Web pages. Thus it answers the needs concerning SweetWiki and Learn-Nett platforms resources that are browsable.
Metadata sharing	Needs an access to ECCO ontologies	Availability of ontologies is already ensured, as they are published on the Web and accessible through a URI.
K retrieval	Could share query related functions: get previous / favorite / most common queries (Interest: Learn-Nett)	

Table 7. List of interactions for **DocReuse** service

Interaction Area	Need or Suggestion	Proposed function
Data sharing	Should have access to documents from production services (Amaya, Limsee3, SweetWiki), for restructuration. (ex: Amaya → DocReuse restructuration service). (Interest: Learn-Nett, Did@ctic, ePrep, Adira)	restructureDoc(doc_uri / doc_content, specific params):

Specific considerations need to be made for the integration of standalone services: Amaya and Limsee3. Nothing prevents them to call Web services provided by the other Palette Services. However, this would be a non-sense to make some of their functions available as Web services, because this would require an HTTP daemon to run when the application is opened, so as to handle requests. In particular for Amaya and Limsee3, retrieving metadata they add is not possible except if

they put it directly into documents or if they provide some independent Web service to be deployed on a Web server, that is independent from the original application.

2.3 Raised Issues

The statements concerning feasibility, requirements and implementation of services interactions introduced in section 5 of D.IMP.03 are still valid. In particular, looking at the needs that have been expressed by CoPs and analyzed in this section, some common issues concerning interoperability are:

- users want to transparently access data, whatever the service used (e.g. documents stored in e-Logbook should be accessible by e.g. Amaya) and wherever the data is stored;
- the transparent access to data means a read access as well as a write access (e.g. editing a SweetWiki page from Amaya);
- users want to transparently access and create metadata, whatever the service used (e.g. documents stored in e-Logbook should be accessible by SweetWiki for annotations) and wherever the data / metadata is stored;
- even if this is still rare in the current scenarios, users may wish to access functionalities of a second service with the data of a first service (e.g. e-Logbook context-aware view used from CoPe_it!).

The first and the second issues are related to the data sharing interaction area mentioned in the previous section. The third issue is related to the metadata sharing interaction area mentioned in the previous section. The last issue raises the concern of *synchronization* of views; that means that if the same data is duplicated in two services to get different views of it; it should be decided whether this data remains automatically updated whenever it is changed by one service, or if the replica is considered as a snapshot that lives its life independently of the original data.

Some of the issues can be summarized as data integration issues which is a classical problem in database research (Halevy & al., 2006), but which is relatively recent considering Web services. That means they concern the problem of combining data from different sources and providing the user with a unified view of these data. For an example of such a common view, we can consider a common search engine for instance. The data integration problem in database research has been solved by forming mappings between heterogeneous data sources and defining one single query interface for all of the sources (Dontcheva & al., 2007). It is still a field of active research for Web services.

The analysis above does not reflect some of the security and privacy issues that have also been very often mentioned during the participatory design process. This task is left as future work for the next version of the deliverable, as there are also a whole bunch of concerns that comes when considering the impact of each service access control policy on the integration of a set of services.

Finally, it is of the uttermost importance that the definition of the interactions between services presented in the previous sections be continued and refined during the elaboration of the generic scenarios in task 4 of WP5.

2.4 Interoperability example

This section proposes an imaginary example, the cross-service document importer, to give a concrete expression to some of the previously defined opened issues. The cross-service document importer illustrates a simple case of interoperability that could be achieved in Palette. It could be generalized to every service that needs to access to document created into another service. It also illustrates a simple cross-service authentication mechanism.

This example has been constructed out of the Did@cTIC scenario. Although it does not appear “as is” in the scenario, it seems realistic and sufficient for our purposes. In this example, a member of the Did@cTIC CoP wishes to transform a document stored in SweetWiki using the DocReuse tool with a document template s/he has uploaded and stored before in his/her private storage space in DocReuse.

The output document will also be stored in his/her private storage space in DocReuse, so that s/he can then download it and send it by email, for instance to other people which are not members of the CoP.

This example illustrates one basic interoperability need, which is to import a document from one tool inside another tool. It is a case of the information exchange level of service interaction.

The figure below shows a possible solution at the User Interface level: a widget box called « Import » is embedded inside the DocReuse Web application to which the user has logged in before (for instance coming from the Palette portal created for his/her CoP). The widget contains a drop down list with the name of all the tools with the capability to export a document available to that user. It also contains a text entry field and an "import" button. By entering a document name and by validating, the user asks DocReuse to request the document from the currently selected tool in the drop box.

Figure 1. Mockup for the selection of a Palette service and of a document to import from another service.

The next figure shows that in response to the user's action, DocReuse opens a modal dialog prompting the user for a login name and password. This is because access to the selected document requires authentication.

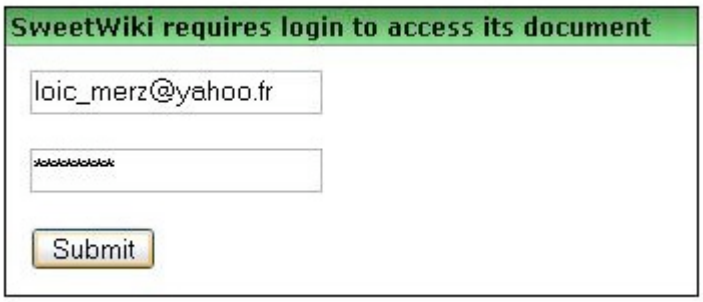

Figure 2. Hypothetical identification dialog inside the DocReuse on request of SweetWiki.

The last figure shows that once successfully logged in, the requested document now appears at the bottom of the "Import" widget, it is assumed from here that it is usable as an input to the DocReuse transformation process which is not detailed here, or that it can directly be opened, maybe with SweetWiki through the provided hyperlink.

Figure 3. Imported document now available in DocReuse.

The cross-service document importer illustrates some of the questions that any Palette architecture proposal should answer:

- how will a tool advertise that it can offer a given service (e.g. document export);

- how will a tool get a list of other tools offering a given service (e.g. document export);
- how will a tool request another tool about a service with some arguments;
- how will a tool respond to a request by requesting authentication of the user;
- how will a tool respond to an authentication request in place of the expected response from another tool service invocation;
- how will a tool get a response for a request sent to another tool?

The cross-service document importer also shows a possible solution at the UI level to integrate access to a remote service of a second tool from a first tool by using a widget composition mechanism.

The next section will survey some software architecture styles that could answer to the above questions and satisfy Palette service interoperability needs. A section suggesting a preliminary solution to start integration of at least two Palette services during the trials will follow it. Section 5 will initiate some guidelines that address interoperability at the User Interface level.

3 Architecture styles for integration of services

The interoperability needs listed in the previous section require that Palette tools conform to some common standards for allowing communication between the tools that will support information exchange, service invocation, or maybe more advanced composition of services if required in the future. This section presents two architecture styles that may lead to successful integration. It starts by proposing a model to define the different kinds of software components, which are designated by the term "Palette service". Then, after the presentation of architectural styles, it is concluded by some reflections about how they fit to the current state of Palette needs and technical achievements.

3.1 Classification of Palette services

Three different main kinds of services can be identified from a technical point of view:

- Tools consisting in software applications installed on the user's machine with their own GUI;
- Interactive Services directly available for the end-user through a Web page;
- Web Services operated on a server and made available only to third-party services.

The distinction between *Interactive Services* and *Web Services* better emphasizes the difference between services associated with a GUI, and faceless services, with no GUI. The main difference is that an Interactive Service is dedicated to human-machine interaction thus providing a Human-Machine Interface (HMI), while a Web Service is dedicated to machine-machine interaction and provides a Machine-Machine Interface (MMI). While the former could be assimilated to a Web application, it also comprises in particular Web Services for which a Web GUI is provided. The figure below gives a complete model of the different services.

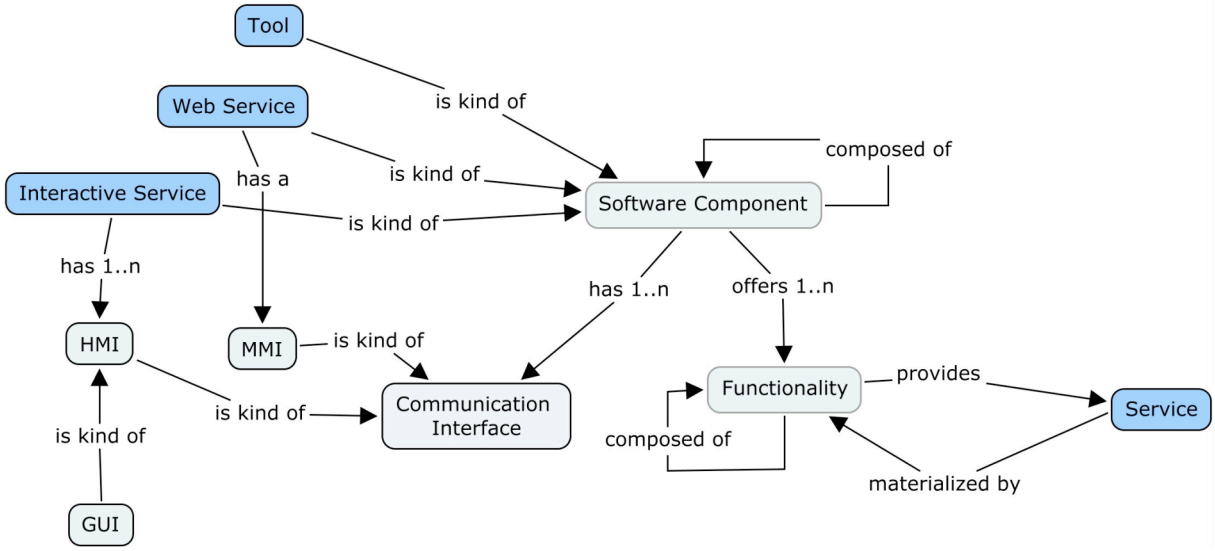

Figure 4. A model defining Palette tools and services.

Summarizing, the different Palette services that need to be integrated can be distinguished by the fact they already offer or not a HMI:

- Web Services:
  - are faceless services with no HMI
  - are functionalities of Web applications also provided as Web Services with no HMI
- Interactive Services:
  - are standalone tools (i.e. desktop applications) or browser plugins
  - are standalone Web applications (with or without extra Web Services offered)

- are Web GUI that provides a HMI to a set of Web Services
- are non-Web GUI that provides a HMI to a set of Web Services such as widgets embedded into a desktop application and that call Web Services

The first role of a common platform would be to offer a simple mean to download and install standalone applications. Such a common platform would also allow to publish a description of all the available services and of the way they can be accessed. Finally it could also be used to allow composition and/or orchestration of Web Services if needed. Concerning Interactive Services, composition at the UI level could be handled if needed following the guidelines for homogeneous UI of the last chapter.

3.2 Service oriented architecture style

The service oriented architecture style (SOA), with the use of atomic or composite Web services, has been first mentioned in the first Palette DoW. This has lead to the definition of a full SOA solution for Palette, which has been presented in February 2007 in a WP5 meeting in Luxembourg (Naudet, 2007). The figures of that section are based on the slides presented during that meeting. Since then, the proposition has been amended and the new proposition will be presented in the next chapter. However, in this section, we reuse some materials from the initial proposition to show how a full SOA could be considered in Palette.

The full SOA solution has the advantage to provide a unique platform as a single entry point for access and management of Palette services by CoPs. The usage assumptions made are the following: Partner tools register their services into the platform; CoPs use the services through the platform. This low coupling between service providers and service users is classical in SOA. It allows users to know the services and not the tools behind them.

3.2.1 Definition

Launched by Gartner Group, SOA defines an applicative interaction model implementing loose coupling connection between different software components. A service refers to an action executed by a provider component for the attention of a consumer and possibly based on another system. According to W3C, a SOA specifies a set of components whose interfaces can be described (WSDL), published (UDDI), discovered (UDDI) and invoked (SOAP) over a network. SOA targets the promotion of software development in a way that leverages the construction of dynamic systems, which can easily adapt to volatile environments and be maintained. The decoupling of system constituent parts enables the re-configuration of system components according to the end-user's needs and the system's environment. Furthermore, the use of widely accepted standards and protocols that are based on XML and operate above Internet standards (HTTP, SMTP, etc) enhances interoperability.

An SOA relies on Web services. However, making an SOA is not equivalent to simply use Web services: the former is an architectural style, while the latter is a technological implementation. Concretely, an SOA can be characterized by the fact it insures loose coupling between services, through a mediator component that will dynamically orchestrate calls: there is no direct invocation from service to service. The next figure shows the constitutive elements of a SOA architecture which are defined in the next sections.

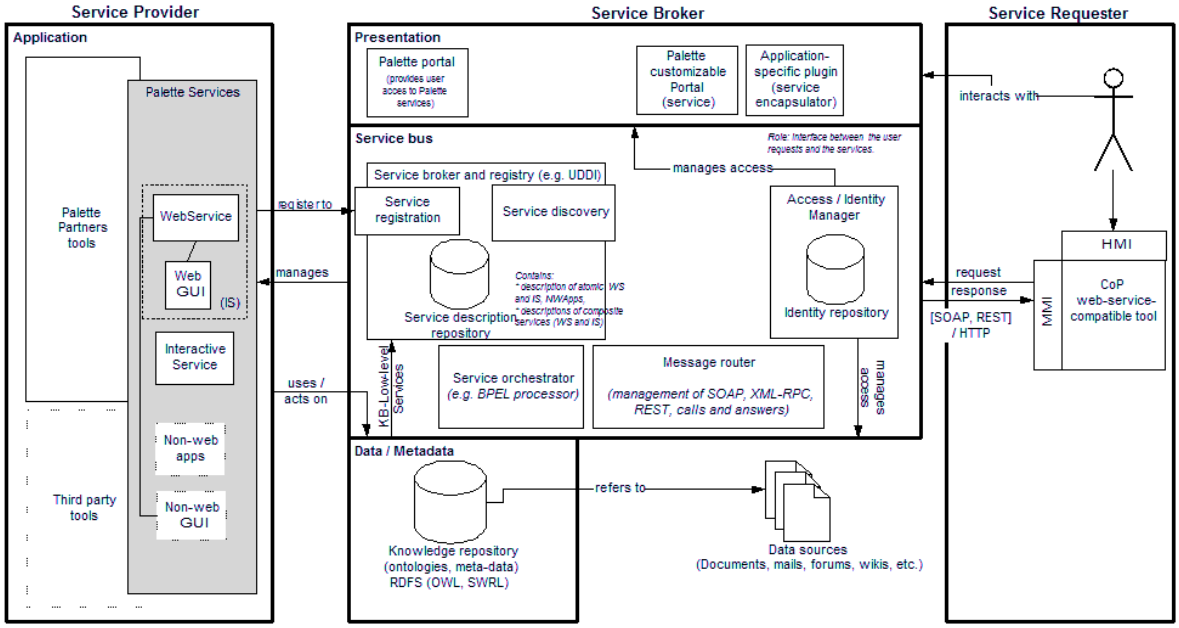

Figure 5. A full SOA typical block diagram applied to the Palette context.

3.2.2 Service Provider Layer

The service providers are all the applications that can act as services provider. The first section of that chapter has defined different kinds of applications that can provide services from standalone desktop applications with rich user interfaces to faceless, non-interactive applications, available only through Web services.

3.2.3 Service Requester Layer

The service requesters are basically represented by entities that will need and use services provided by the platform. Different use cases of service call can be considered:

- the requester accesses a Web service from an application (machine call);
- the requester accesses a Web service using the service dedicated Web GUI via an URI or via a specific component inserted in an application;
- the requester accesses a Web service using a non-Web GUI via a specific component inserted in an application;
- the requester accesses a interactive service via an URI or via a specific component inserted in an application;
- the requester executes a non-Web application.

3.2.4 Service Broker Layer

The service broker is itself constituted by several layers: the Presentation layer, the Service Bus layer (SB), and the Data / Meta-Data layer. It plays the role of middleware between providers and requesters by being:

- a single entry point to access services → registration service for providers and portal for requesters;
- a matching requests & abstract descriptions to services → discovery service;
- a message routing between requesters and services, and between services → message router;
- an orchestration of complex services → service orchestrator;
- a service execution engine.

The Service Bus

The main aim of the SB is to handle requests and to route messages between services requesters and providers. It comprises a service orchestrator allowing to manage and execute composite services. Typically, it can be implemented using an Enterprise Service Bus software. For instance Mule (URL: MUL) could be used, even if no choice is necessary yet in the Palette context as it will be described in the next chapter. The SB plays a role of mediator between service consumers and service providers, solving point-to-point integration issues and insuring a loose coupling between applications providing services, and between services. Every service request from a consumer is mediated by the Service Bus, which localizes service providers and routes the request. At this stage, diverse transformations might be required, at the communication protocol level or message level.

The roles given to the SB are the following:

- being a registry of services, offering service registration and discovery functions;
- orchestrates service composition needs, in particular with the handling of composite service scripts;
- routes the messages between requester and provider, and manage transformations in communication protocol, message syntax and semantics (e.g., handle SOAP/WSDL and REST/WADL interchange, provide adaptors for different XML languages (XSLT transformations);
- being a single entry point, managing authentication, access and security.

As shown on the following figure, four components correspond to the four aforementioned goals: the Service Broker and Registry, the Service Orchestrator, the Message Router, and the Access/Identity Manager.

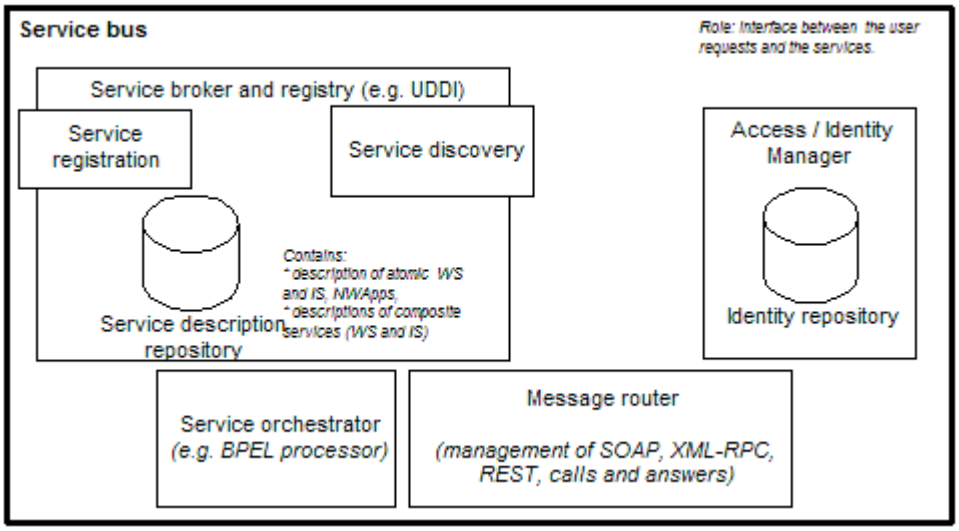

Figure 6. Service Bus Architecture

The Service Broker and Registry component is composed of the Service registration, the Service discovery and the Service description repository. The service registration manages registrations of services and stores services descriptions into the repository. The service description repository then stores descriptions of registered simple and composite services, those descriptions containing both human- and machine-dedicated information. The service discovery matches queries to services descriptions in the repository and returns descriptions usable for calling the service (e.g. WSDL, WADL, BPEL, etc). The service discovery has a central role in the SOA architecture, i.e. it is not a simple search engine. It is used for matching requests coming from any requester (users, applications, services) and for automatic discovery of services (semantic descriptions could be used here).

The Service Orchestrator interprets and executes composition of services. In this case, the broker is systematically the unique interface with the requester and no direct service call is usually possible, except if the requester possesses its own service orchestrator. In the case of Web services, a composite service can be described using e.g. BPEL or WS-CDL for Web services, depending on which kind of composition occurs: orchestration or choreography. Additionally, we can have also considered composition of Interactive Services. In this case, however, a graphical composition model must be defined, accounting not only for graphical composition but also for the event managements between considered services. Whatever the kind of services composed, the Service Orchestrator processes the description of the composition, finds through the Service Discovery atomic or composite services required, and executes the composition handling the workflow, the connection to services, the errors, rollback needs, etc. Finally, the Service Orchestrator manages the transactions with services providers and maintains the links with the latter, in order to be able to redirect to similar service on another provider in case of provider deficiency (redundancy between services is good!).

The Message Router handles external and internal messages by routing them and transforming them if the input or output is in a different format. The technologies handled could be SOAP, REST, XML-RPC for Web services calls, RDFS or OWL with a RDF/XML syntax, or a N3 or N-triple syntax for description. However the last two syntaxes are not standard and should not be preferred.

The Identity Management and Access Rights component is used to manage identity and access of users. The Identity Management module can be centralized with for instance a LDAP server use for all services. But if the set of services is possibly open, we can foresee to use decentralized systems like OpenID (URL: OPI), Liberty Alliance (URL: LIA), or CardSpace (URL: CAS). However the last one is a proprietary solution that should not be preferred. Once a user is identified on the platform, the platform enforces his/her rights. As each service needs to manage its own rights we need to standardize users' profile. The single sign-on principle should be used to avoid that users connect to services by entering login and password several times as we have shown in the imaginary example at the end of the first chapter.

The Presentation layer

The presentation layer of the Service Broker layer offers components from which users are able to interact with the platform and its services.

For instance a portal component can provide a unified Web User Interface to service management and service use. Among others, it can provide the following classical functions:

- registration, discovery, browsing, direct use of services (requester and provider access);
- management of data, metadata, services, users etc (administrator access).

Portals offered by Google or Netvibes are examples of customizable portals of that sort.

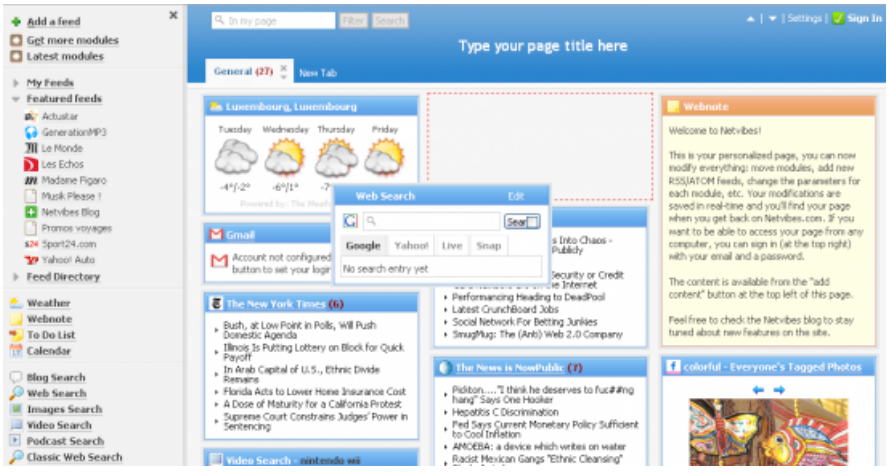

Figure 7. A well-known customizable portal, Netvibes.

The (meta-)data layer

The optional (meta-)data layer offers a (meta-)data registry or repository service, referencing or storing knowledge in a canonical form to facilitate consistent access and processing by services. Its role is to support the access to distributed and non homogeneous (meta-)data storages, handled by different services and tools.

3.3 Data mashup oriented architecture style

The data mashup oriented architecture style comes from the emphasis on building interactive applications by aggregating and integrating third-party data from multiple sources (Liu & al., 2007; Erenkrantz & al., 2007). Its main characteristics are:

- "static" Web pages are replaced with more "dynamic" interactive pages that consume data produced by third-party applications (by RSS, Atom[1] or REST);
- data contained in a Web page is organized in a componentized manner (i.e. the atomic element of a page are Widgets in place of basic DOM - Document Object Model - elements);
- the consumers (i.e. the clients in client-server terminology) can even accomplish some business logics in the Web browser instead of accessing the layer residing in the server side.

The condition of application of data mashup architecture is to build RSS/ATOM/REST support in the services that aims at being consumed to create mashup applications.

3.3.1 Mashup architecture

A Web mashup is a Web application that combines data from two or more external online sources. The external sources are typically other Web sites. Their data may be obtained by the mashup developer in various ways, including, but not limited to APIs, XML feeds, and screen scraping. The later one is a technique that involves parsing the human oriented output of a service and converting it to data. A Web mashup access to a minimum of two data sources that can be combined to create a service, which is not otherwise available from either source.

Mashup consists of the following three entities:

- A Web site, the mashup host;
- One or more servers, the content providers;
- An execution environment, usually a client Web browser.

The figure below shows the mashup architecture. The mashup host generates an "active page" for the execution environment (client Web browser). The active page contains references (URL) to resources maintained by content providers and a set of client-side scripts in JavaScript. The client-side scripts, executing in the client, interpret user actions and manage access to, and presentation of, the combined representations served by the content providers.

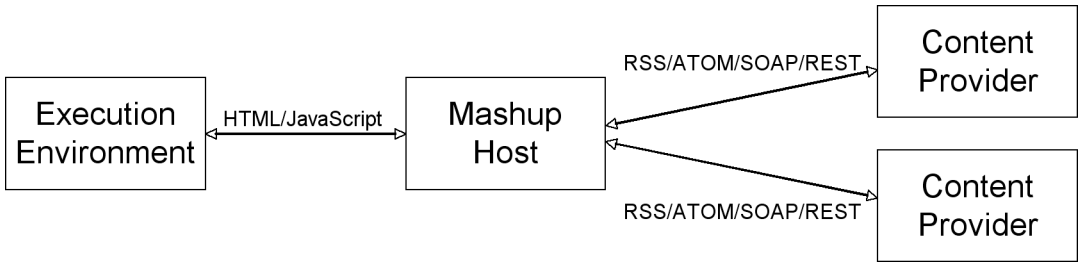

Figure 8. Mashup architecture.

In a data mashup architecture the role of the mashup host is limited to construct a virtual "redirection" comprising a set of client-side scripts that reference resources hosted elsewhere at Web sites (content

[1] Atom is an alternative document format to RSS.

providers). Thereafter, the client interacts only with the content providers and sees the mashup host as a proxy and as a data aggregator. The presence of the mashup host is also a convenient way to bypass the browser *same origin* security policy that usually prevents a Web application executing inside a browser page to access services not originating from the same origin as the page.

The mashup host "synthesizes" a virtual compound resource for the client. Though a mashup could be implemented entirely server-side by the mashup host, it would increase server load and (for highly interactive mashups) latency. Mashups illustrate both the power of combining multiple resources under computational control to synthesize a new resource and the benefits of moving the locus of computation away from the origin server.

3.3.2 Mashup example

There are many examples of mashup applications. A popular application is to combine location data (such as housing, travelling) with geospatial data service (such as Google Maps, Yahoo Maps). Some mashups offer product listings, ratings, auction prices, and so forth by combining catalog data from Amazon with auction data from eBay.

Several mashup editors are provided for developers to build mashup applications. One of the popular editors is Google Mashup Editor (URL: GME), Yahoo Pipes (URL: YPI), and Microsoft Popfly (URL: MPF). In this section, we reproduce a mashup example built by GME as given in its presentation by Google.

The left side of the figure below shows a conference travel application. It is a Web application that pulls events off a calendar and plots the event location on a map.

The right side of the figure below shows the data and control flow of the application. It displays data (also called the « feed ») from Google Calendar in a list module. Every time the user selects an item in the list, the application updates the item module just below the list. It also triggers a short JavaScript function, which displays the event location on a map using the Google Maps API.

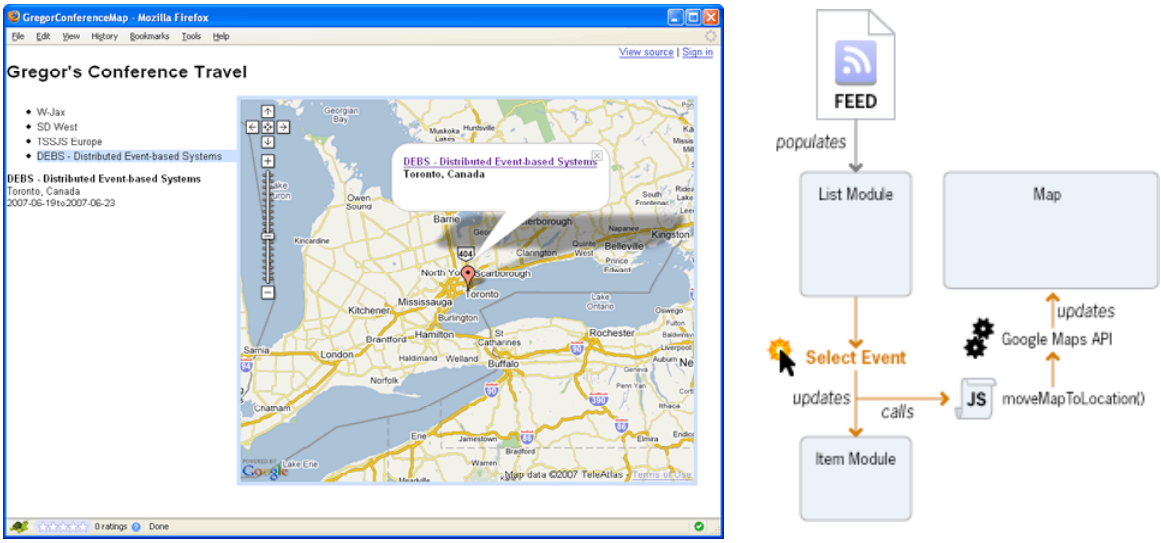

Figure 9. Conference travel application, and its data and control flow.

3.4 Lessons learned from interaction with CoPs

As mentioned in D.IMP.01, the Palette project aims at offering services to be adopted by a wide range of CoPs; for this purpose, services to be provided need to be as generic as possible, while accommodating features that may drastically differ from one CoP to another.

3.4.1 Centralized versus distributed execution

One of the challenges on the technical side is to propose an adaptable and scalable platform of services that may be enriched over time. In particular, one of the most important issues concerns the location of the software code that will be deployed and run for operating Palette services.

The study of the usage scenarios, started in D.IMP.03, allows to further elaborate technical guidelines on this aspect, and to choose some technical options that underline the design of the global software architecture for delivering Palette services. Indeed, presently each service is not supposed to communicate with all the others, but instead it mainly interacts with a relatively small subset of the whole range of services, thus creating groups of interacting services (or in other terms: strongly connected sub-graphs in the graph of interaction of services). This is because a user typically interacts with a main service enriched with features offered by a few other services.

The previous observation could lead to the conclusion that there is no need for a centralized authority to manage communication between services and that peer-to-peer communications between groups of services could be sufficient. However, this is not so obvious, as groups of services expected to tightly interact with each other are very different in nature. Thus taking the intersection of all the groups of interacting services could very easily lead to a global interoperability level sought in Palette as new interactions between services are requested by CoPs. This reasoning could lead to the conclusion that there is a need for a centralized execution model at least for mediating communications.

There are also two types of arguments in favour of a high level of flexibility in the execution model of services. The first kind concerns usability and the second extensibility.

From the point of view of usability, the services differ greatly because of their required execution environment. Some services may rely on powerful authoring environments in which it is crucial to provide the user with very low time delay when responding to his interactions. Since this kind of service typically runs rather costly incremental formatting algorithms, code installed and executed locally - and independent from any connection - is most appropriate for guaranteeing a satisfactory level of usability. Examples of services offering such environments include Amaya and LimSee3.

On the opposite, some services gain or assume to be operated with their main code running on an independent server. For example, awareness-related services (such as e-Logbook) rely on the permanent online availability and reliability of a server for delivering notifications to a variety of user machines and portable devices.

Finally, some services may greatly take advantage of having their main code deployed in a well-balanced manner between a server and user machines. Many services can be made (at least) partially available from a variety of small devices. For example, a mobile phone could be used to issue a query to a semantic search engine operated on a remote server. Such kind of services may deploy appropriate user interfaces (such as widgets) on the client side for providing the user with possible interactions that are adapted to the context.

From the point of view of extensibility, the addition of new services is bound to the interesting consequence that new - and a priori unknown - features can be obtained by composing features from different services. Last but not least, the platform should allow for seamless integration of updated services as well as future services. Whenever a new service becomes available, the impact of its deployment should be limited. For example, it should not require the redeployment of all services, but instead at most the update of the highly related services, if needed.

As a consequence, the global architecture should allow a wide range of options concerning code location. In particular, it should not enforce a centralized or distributed execution model, but it should allow the cohabitation of both models as software code operating Palette services is necessarily variably distributed between servers and user machines.

3.4.2 Suitability of a SOA architecture style in the current Palette context

The SOA architecture style has low coupling between components which is a great advantage for extensibility:

- Providers know the Broker → Partner tools register their services into the platform
- Requesters know the Broker → CoPs use services through the platform
- Requesters do not know the Providers → CoPs do not know the tools

On the other side, the SOA architecture style requires some centralized running infrastructures so as to access the services and to allow communication between services or between services and requesters. It also requires the definition of common generic application programming interfaces (APIs) for all the services if they are to be interchangeable.

The analysis of discussions with CoPs, reflected in the current proposed scenarios, highlights that the current situation, already stated in D.IMP.03 has not evolved: CoPs use mainly "big" Palette services offering the maximum of function they need and occasionally use functions offered by other Palette Services for specific processing.

In Palette, the needs and the objectives of CoPs have been elicited following a participatory design approach. The identification process should as far as possible be independent from any known or expected solution. However, to bootstrap the process some subsets of available tools and solutions have also been matched with the corresponding subset of problems that happened to be found a posteriori in the CoPs, and have been presented to the CoP members.

The set of existing tools available during the identification of the needs had significant consequences on the service nature and architecture. The collaboration between CoPs and Palette partners that has lead to the scenario proposals and the development of some Palette services has been influenced by the existing tools that have been proposed to CoPs and further enhanced to answer their needs. This has lead to the following consequences:

- Services identified on the basis of isolated tools leads to the same services being identified by different groups with different names. As a result, no common or shared services are identified. This is a typical anti-SOA approach, which is sometimes called the Silo Approach (URL: TSA).
- The approach has lead naturally to requirements for local interactions rather than global ones. The direct consequence is that proposed Web services are not sufficiently fine-grained to make SOA a suitable solution, additionally because dynamic routing of messages and service orchestration is not needed.
- Finally, a direct consequence is that interoperability between Palette services is seen from local groups of services and not from a global perspective (to all Palette services). SOA is suitable when a global view is taken, which is not the case here.

In conclusion, SOA is suitable for loose coupling between services that can be interchangeable and for complex service composition. In the present situation it seems that some features of the SOA architecture, such as the Message Router in the Service Bus are not needed to route messages between requesters and providers, as in the scenarios of interoperability they know each other in advance.

3.4.3 Suitability of a data mashup oriented style in the current Palette context

The data mashup style is adapted to distributed client-server systems where the services are easily exposed as resources, which can be manipulated through REST requests as defined by (Fielding, 2000). It works well if the services form a distributed hypermedia system where the nodes provide multiple sources of content. Then it is a solution which can be lightweight in implementation and built with a minimal amount of code.

Compared to current SOA composition technologies, such as BPEL and WSCI (Web Service Choreography Interface), mashup is more "coarse-grained" at the application level. Indeed, each mashup application is a combination of data, processes and user interface with its own context and business logic. Therefore, each mashup application embeds its own scripted composition, or choreography, of services, which is less easily reusable than a composition or choreography defined in a declarative way.

According to the possible interactions between Palette services described in the first chapter, most of the interactions consist of two Palette partner tools. The data flow of the interactions is mainly: data is retrieved from a source application, the data is transmitted to a destination application, and finally the data is processed in the destination application. These interaction examples include: (Amaya → e-Logbook), (LimSee3 → e-Logbook), (Sweetwiki → e-Logbook), (CoPe_it! → Sweetwiki), etc. These service compositions do not involve complex business logic as most of the time the data produced by a source is simply fed into another service. As mashup is useful for small and simple application integration logic, it might be an alternative to more classical approaches for service composition in Palette.

The figure below shows an example of a mashup architecture. The content providers are the Web applications that provide data in the format of RSS/Atom/SOAP/REST. Mashup applications are built according to use cases and scenarios. Each mashup application retrieves data from one or more content providers, integrates the data, and presents the integrated information on the user interface of the mashup application. An application portal can be used to list the available mashup applications. Consumers can execute the mashup applications through their client Web browsers.

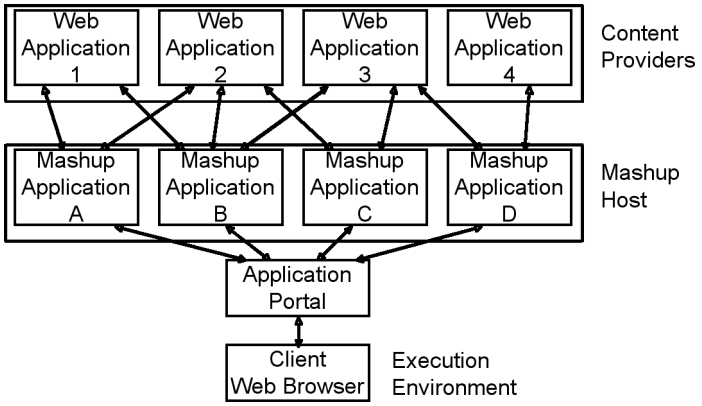

Figure 10. Example of a mashup architecture.

The requirements for deploying a mashup architecture are as follows:

- *Mashup hosting site*: the mashup host needs to provide modules for data manipulation, event handling, and user interface. A mashup editor is helpful for the developers to develop mashup applications.
- *Communication protocol*: mashup is a Web-based service composition. The content providers need to support Web-based communication protocol including either of RSS, Atom, REST, or SOAP.
- *Data consistency*: the content providers need to achieve coherent data representation on data. For example, they need to represent the data of "file modification time" in same format.
- *Authentication*: if content providers require user authentication, since mashup applications involve access of multiple content providers, they need to have access rights to access the content providers.

In conclusion, mashup may be suitable for simple service compositions based on content aggregation. It requires that a different mashup be programmed for each case of composition identified in an ad'hoc manner. The aggregation is from the content providers to the clients, but nothing prevents the clients from sending back data to the content providers or to update it with explicit requests.

4 Towards a Palette Service Platform Architecture (PSPA)

4.1 Rationale

The proposition of a Palette Service Platform Architecture (PSPA) must take into consideration both the analysis of the Palette scenarios (see Section 2) and the initial SOA proposal made in the first period of the project. It must also take into consideration the latest achievements in terms of architecture styles, in particular the fact that the data mashup architecture style can now offer interesting alternatives to achieve some forms of interoperability (see previous section). This has lead to numerous discussions that have taken place in the project's technical meetings and between WP5 partners. From these discussions, it came out that at this stage of the project we still want a platform allowing to experiment with both the mashup data architecture style and with the service oriented style.

For these reasons, the architecture that is suggested in this section has been inspired by the classical SOA architecture, but it has been approached through the current status of the project, the CoPs' requirements and the related discussions. This architecture is fully compliant with the tasks of WP5, and it will allow partners to develop in the next months "proof of concepts" implementations of different styles of interoperability (some more simple without scripted composition of services, while more advanced styles are still possible). Moreover, the architecture can accommodate different software patterns to expose the services such as the resource-oriented pattern with REST, which will be preferred, or the activity-oriented pattern with SOAP (Snell, 2004). This is illustrated through an example given at the end of the section.

4.2 Suggested conceptual architecture

The goal of the suggested conceptual architecture is to support a set of useful services to the end-users of the framework:

- the provision of a Web based visual Palette service integrator, with which end-users will be able to build individual working environments;
- the registration of services through the creation, manipulation and the publication of the Palette service description records from the Palette service providers through a secure way;
- the creation of new composite or orchestrated services, if this becomes a necessity in the forthcoming generic scenarios.

For this purpose, PSPA is defined as a set of modules adapted from the SOA layers: Service Provider, Service Broker and Service Requester. The service providers and service requesters are preserved unchanged. The Service Broker layer is transformed into a Palette Service Registry and Delivery layer. The latter layer still conforms to the specifications agreed in IP2 (Marache, 2007), even if it does not retain the full complexity of a Broker layer. The next figure gives an overview of the platform, with the Palette-specific Broker layer in the center.

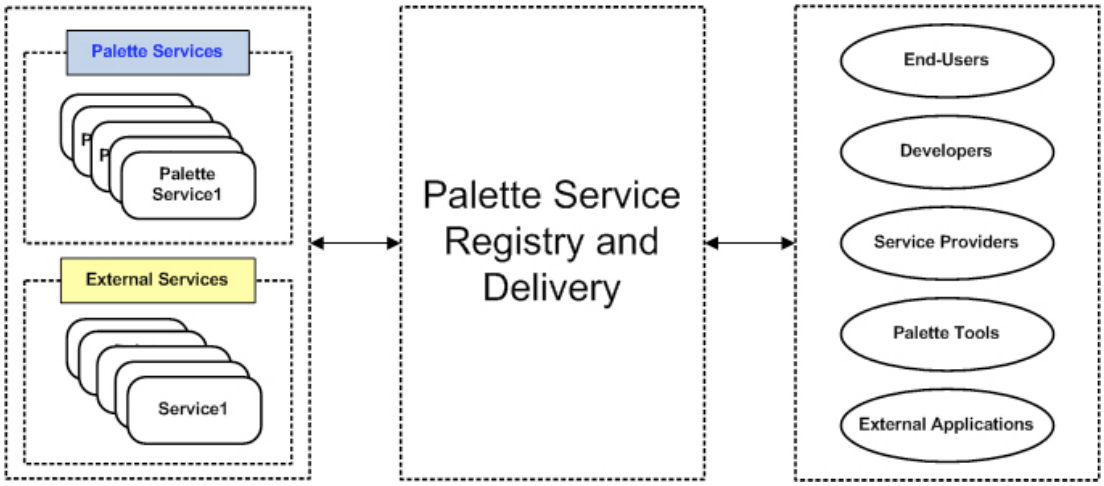

Figure 11. An overall view of the platform: in the left, Palette services are offered by service providers to service requesters (right) through the Palette Service Registry and Delivery (that would correspond to the Service Broker layer in a pure SOA style).

PSPA is described in the following order in the next subsections: the services provided at the Service Provider layer are described first, followed by the users of the platform at the Service Requester layer. These two layers are similar to their counterparts in the classical SOA architecture style. Then, the next three subsections present the specific Palette modules that constitute the counterpart of the Service Broker layer: first the Palette Service Registry Framework, then the Service Orchestration module, and finally the Palette Service Delivery. The last one corresponds to a Presentation layer and comprises two modules: the Registry User Interface and the Palette Portal. All these modules are depicted in the next figure from a design perspective.

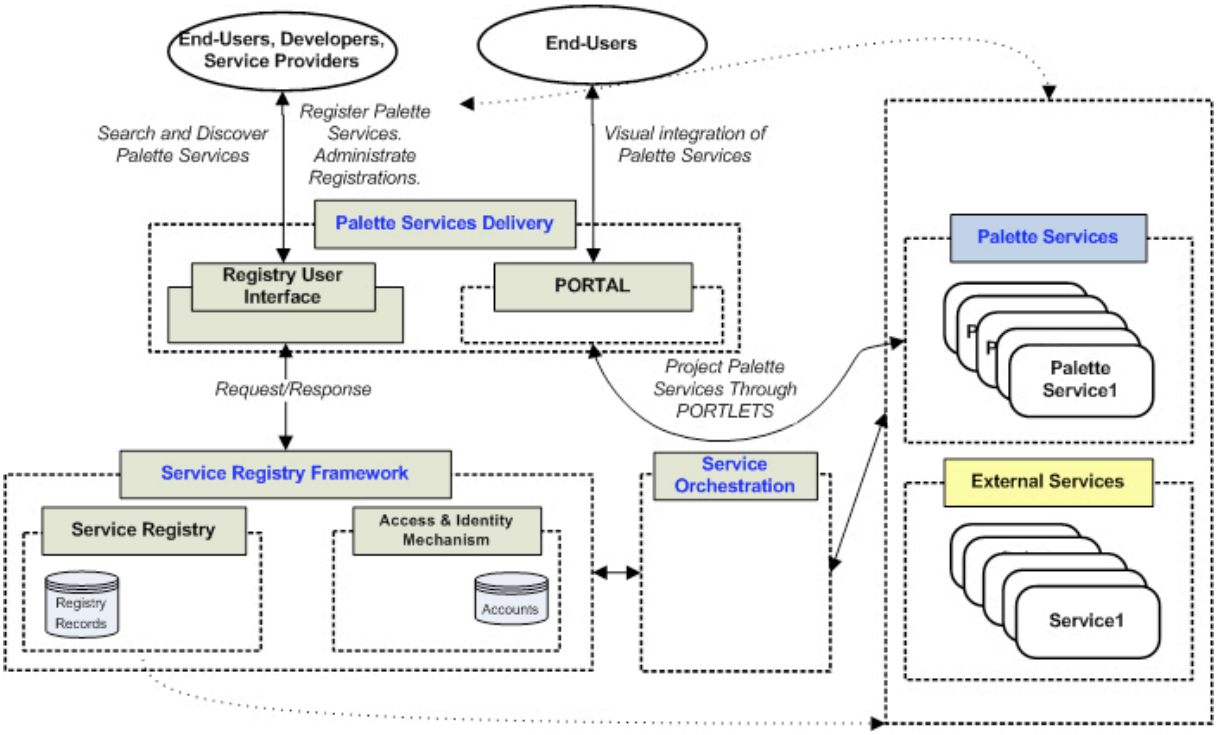

Figure 12. The overall architecture from a design perspective.

4.2.1 Services supported at the Service Provider layer

In the rather complex context of Palette, services provided are either faceless Web Services or Interactive Services as defined in subsection 3.1. *Third-party tools* can also be considered as candidates to provide services useful for CoPs.

The diversity of the provided services is a critical parameter to be taken into account during the design and implementation of the proposed system. In particular, the current platform has to support the registration, discovery and delivery of all kinds of services provided by the Palette partners as well as external service providers. In this context, Palette partners have to comply with a set of guidelines that emerged through various WP5 meetings. Thus, each Palette service or tool has to be available (for access or download) via a URL, regardless if this service is a Web Service or an Interactive Service. Furthermore, for each provided service through the framework, a service description record has to be available with both human- and computer-interpreted information. Finally, the service providers have to be responsible for the registration, update and publication of their services.

The initially supported services for the framework are classified into two categories: the Palette services, and the external services. Palette services include the variety of services defined in subsection 3.1, while external services contain a set of some useful services that are already implemented outside the Palette project. Such services may vary from instant messaging services (e.g. Jabber server and/or clients) to geographical services (e.g. Google maps), and real-time video or streaming services. Although the suggested communication protocol for the provided services is REST or SOAP (for enabling easy interoperation), it is obviously possible that many supported services may operate under different protocols. In any case, the proposed framework can support a diverse set of services.

Finally, composite services are another type of services that may be offered in the future if required by new scenarios. Composite services may derive from either the composition of the existing Palette services or the composition of external services or services not registered in the Palette context. Furthermore, it is possible to create composite services through the service orchestration mechanism as described in WP5 of IP2.

4.2.2 Users of the Platform at the Service Requester layer

Palette services can be used by potential users – both human and machines - classified into the following categories:

- **end users,** such as CoP members, will have the ability to select and use the appropriate Palette services according to their needs;
- **developers** inside or outside Palette scope, may search and locate interesting services to integrate into their applications. Furthermore, developers may have the ability to create new or composite services out of the existing ones and make them available through the Palette Service Registry;
- **Palette Service Providers** will use the infrastructure in order to advertise the Palette services to the world;
- **Palette Tools** that want to access the provided services in design or in run-time may be potential users of this infrastructure as well;
- **external Applications or Services** may also access the provided Palette services for the same reasons.

In the last two cases, depending on the state of development and the tools and applications considered, the PSP's role can be limited to locating the services, the services then directly addressing each other in a mashup, or peer to peer, oriented way, or the services can use the orchestration module of the PSP as a mediator to call each other. In the last case, they could benefit from additional services such as ontology-based transformation of requests.

Whatever the role of a CoP member is, user interactions within the PSP happens at the Presentation layer, which can contain different UI as described below.

4.2.3 The Palette Service Registry Framework (PSRF) in the Service Broker layer

PSRF is responsible to hold and publish description records of all Palette services. It is available to developers during design time, and to end-users as services are deployed. To facilitate the service discovery process, service descriptions records are being constructed so as to include both human- and machine-readable information, as well as to include a set of metadata such as keyword, descriptions, etc. with a semantic dimension. The main tasks of the PSRF are the following:

- to publish the services by registering service descriptions to the registry;
- to modify service descriptions;
- to search and discover services;
- to handle client authentication.

Internally, PSRF is divided into two different sub-modules: the Service Registry and the Access and Identity Mechanism.

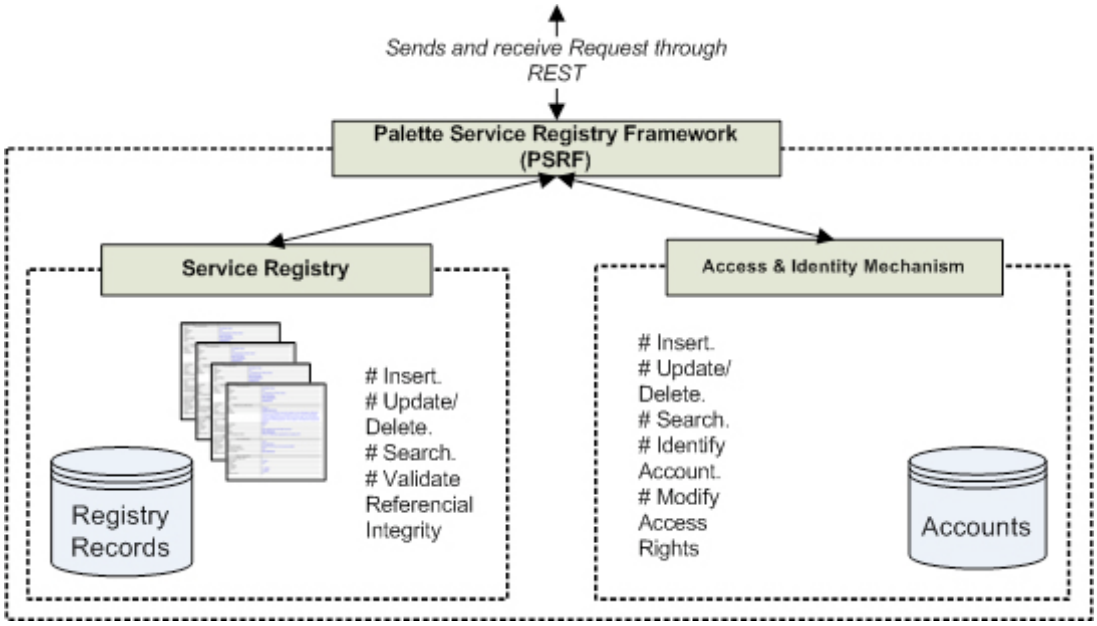

Figure 13. PSRF internal modules

The Service Registry handles all the operations concerning the management of the service records, while the Access and Identity Management is responsible to authenticate users and applications. Both modules are using a relational database as storage means and are closely coupled in order to provide a secure framework in which authenticated users can store, manage, publish and search service description records.

The Palette's Service Registry concerns the mechanism that holds the descriptions of all published Palette services. In particular, the requestors are able to search, locate and finally use the Palette services descriptions during design time in order to generate the necessary "stubs" for communicating with other Palette services and integrating them with their own. Those descriptions (registry records) are stored internally in an XML-format that is compliant with a specific Palette Service XML Schema. Palette Service provides to the PSRF API a rich set of operations that can store, manipulate, publish and search Palette Registry Records.

With respect to Palette's IP2 (WP5, Task 2), PSRF is a Web service itself. It is a stand-alone server that operates by sending and receiving XML-formatted messages over the HTTP protocol.

The Access and Identity Mechanism module is responsible for managing the identity repository and for executing the required mechanisms that will authenticate users and applications. The main entity of this module is the identity entity called "Account". Accounts can be related to humans or applications.

In particular, PSRF supports currently three different types of accounts: Administrator, Service Provider and Simple User or Application account. When someone is being authenticated as an Administrator, all the PSRF API methods become available. Service Providers can create and modify registry records into the framework, while simple users (or applications) have view and search privileges only.

4.2.4 The optional Service Orchestration Module in the Service Broker layer

It is responsible for the mechanisms that capture declaratively the interaction between already published services in order to be used and executed by orchestration engines, thus allowing the creation of composite services. This module will only be considered if the needs of the CoPs evolve in that direction. Subsection 4.6 analyzes some of its characteristics.

4.2.5 The Presentation layer of the Service Broker layer

The Palette Service Delivery contains the graphical user interfaces of the PSP. The main goal of this subsystem is to successfully deliver Palette services to CoP members, as well as to provide a GUI for publishing, managing and discovering Palette services.

Through the Palette Service Delivery, the service composition and aggregation is achieved. A single entry point for CoP users is a composable Portal that will be usable by CoPs to construct their own tools by combining different portlets giving access to Palette services. A second entry point is the Registry UI that is a GUI to the PSRF. These are described below.

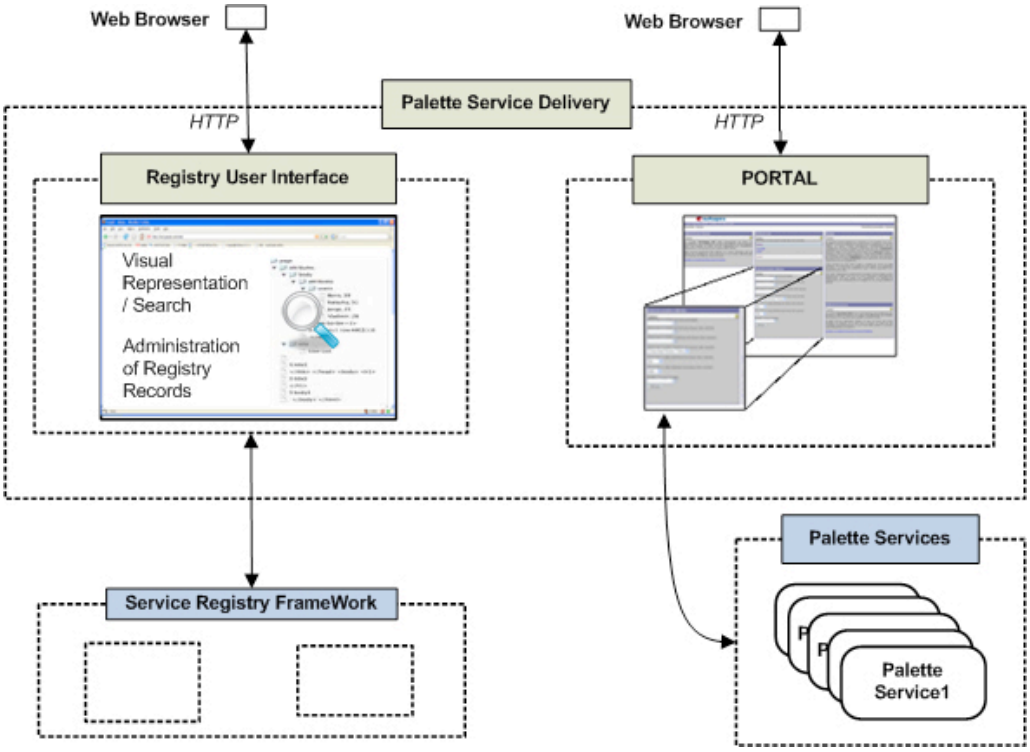

Figure 14. The Palette service delivery subsystem.

The Palette portal

Requester users access the Palette platform through a customizable Web portal, which is a Web application offering a full customizable tool, aggregating Palette services chosen by the user (the user composes his/her own interface). There can be a portal for one CoP or a portal for each CoP member.

The Palette customizable portal is a specific Palette Service. It is a Web application offering a full customizable tool, aggregating Palette services chosen by the user who thus composes his/her own interface for accessing the CoP Palette services. This portal, providing access to a selection of registered services (either Web services or interactive ones), is dynamically customizable either for the

whole CoP or for its members. Typically, it will present multiple Widget components giving a view on services. It could also propose multiple tabs for multiple views (one view for the entire CoP and specific views for the members of the CoP). The user searches or navigates through available Widget components and then adds and arrange the chosen Widgets in the portal. The technologies that can be used are XHTML and CSS. The portal should allow some forms of interoperability between different technologies through the definition of adequate Widget components. Portals offered by Google or Netvibes are examples of customizable portals.

The Registry User Interface

The Registry UI (also known as the Palette Search Engine or Palette Browser) is a graphical interface enabling CoP users and/or developers to search and use new services in the Palette services registry, as well as service providers to publish and manage their service descriptions in the PSRF. It will be developed in Task 3 of WP5. Depending on the UI design choices, it may be proposed to directly develop the Registry UI as a set of specific widgets for the composable Portal.

Implementation guidelines

The two subsystems will be released as Web applications in order to allow accessibility with minimum prerequisites. Regarding services aggregation into the composable Portal, widgets with well-defined specifications will be available in order to include Palette services and then display them into the composable Portal. The Registry UI will communicate with PSRF through its API via REST protocol.

Other GUI

Application specific plug-ins might be developed to allow the use of Palette services directly from a specific application. For example, LinkWidget is such a case of plug-in, developed for Mozilla Firefox. The inverse case might also be considered; for instance, a plug-in that would send some output of a specific host application directly to a Palette service that can handle it.

4.2.6 The CAKB in the (meta)data layer of the Service Broker layer

The Cross-Awareness Knowledge Base (CAKB) has been first proposed during the Palette Developers' Meeting held in Patras, Greece, 26-28/02/2007:

> "This knowledge base would give cross awareness of activities within the CoP, an overview on all resources (pieces of information: data and metadata) whatever their location, whatever which tool manages the content. Those resources will be structured according to ontology models defined in Palette (WP3)".

The CAKB has not been drawn on the previous diagrams of the PSPA, but according to the SOA architecture definition in chapter 2 it would take the place of the (meta-)data layer of the Service Broker layer. We propose to divide the envisioned functionalities of the CAKB into three parts described below.

First, it is a registry of resources and their annotations circulating in the various services and tools of a CoP. As a registry, the CAKB stores links to resources and annotations on these resources. This aims at recording all information, relevant if possible, about the work of members of a CoP on different tools. A storage space is made available in an optional way for services, which want to share their resources on a central repository to avoid duplication and possible problems of synchronization.

Second, an interactive service allows an animator or member of a CoP to search through the various resources managed by the CoP tools and to select resources or annotations that are relevant for a certain task. Moreover, this interactive service can provide a global overview of the CoP activities, by aggregating data and providing adequate graphical representations. The information stored in the CAKB can also be used for statistical purposes by other tools or services of the CoP.

Third, the CAKB can provide some low level support to a service for identifying which services to call in order to get resources, which meet certain criteria. In particular, the CAKB can be a bridge for the data transfer from one service to another, or it can provide the necessary information to a service to establish a peer-to-peer link in an automatic way. However these features are still under discussion as

they may as well be implemented by other means (i.e. a real service discovery module and a real message router module which have currently not be retained in the PSPA that does not define a full service bus in the SOA meaning).

4.3 Communication protocols for interoperability of services

Most of the Palette Services will operate as Web services and communicate with the world through REST or SOAP protocol. Some technical discussions have also led to the addition of the RSS protocol to provide data sources and notification event lists that may be aggregated into other services, and to use SMTP as a means to provide low cost and ubiquitous (aka email) UI to some services. Moreover, in some cases of using external services, it is possible to use alternative protocols (like Jabber, streaming protocols etc.) for achieving specific functionality.

Concerning the PSRF subsystem, PSRF provides an API to potential clients, with which the PSRF remote management is possible. The communication protocol of the PSRF is the REST protocol. In particular, PSRF "listens" to HTTP POST requests that contain XML based messages of predefined type in the BODY section. Then, it replies with an appropriate XML based content into the BODY of the response.

The Palette Service Delivery operates as a Web application and can be accessed through HTTP via common Web Browsers. Internally, this subsystem communicates with PSRF through REST and integrates the available Palette services through well specific Widgets. More specifically, the Registry UI will access the PSRF API through REST calls to provide all the necessary functions for user registration, service registration, access, search, etc. Concerning the Palette Portal, it will be directly connected to Palette Services via widgets developed for them. The communication in this case will be direct, also using calls on available RESTful services.

Finally, the service orchestration engine will handle a composition of RESTful services provided by Palette Services. While the composition itself will be described using BPEL, WS-CDL, or a dedicated XML language, composite services will provide a WADL or WSDL interface allowing communication through SOAP/HTTP or REST/HTTP, respectively.

4.4 Scenarios of usage

- **Registration and publication of a new Palette Service**

Whenever a Palette partner creates a new Palette service (the service is ready to be published to the Palette Community), the following procedure is following:

First, the Palette partner has to be registered itself to the PSRF in order to acquire a Provider Account and obtain the appropriate privileges for creating and manipulating Service Description Records. This task can take place through the Registry UI of the Palette Service Delivery.

After that, the Service Provider has to create a Palette Service Description Record based on the predefined XML format that is already described. The service description included in the Record will fully describe the provided service and will concern both human and machine understandable information. An access path or a URL for download of the service is required for each service description.

The Palette Service Provider will then have to create a new Palette Service entry in the PSRF and fill in all the required information that will be prompted by the Registry UI.

Finally, the Palette Service Provider will be able to modify the Service Registry through the same interface and make it public or not to the Palette project.

- **Service discovery (all kinds of services)**

The Registry UI of the Palette Service Delivery provides a Web-based service discovery through which potential users can search for appropriate services according to their specific needs. Some search criteria that can be used in the search form are: the type of the service (mediation, argumentation etc.), the service provider, general keywords, title, last published services etc.

After a service is found, users can view the description by accessing its Registry Record. Service descriptions can be used for provision of both information of the readers on the service operations and navigation information regarding the location where a service operates or is available to download.

- **Service invocation**

In order for a requester to invoke a Palette service, both the service type and access information that are presented in the service description have to be provided. Depending on the Palette service type, service invocation may take place either through direct communication between the requestor and the service (in cases of existing individual services) or through invoking the Service Orchestration module in cases where the latter is responsible for the execution of the composite services. Requesters are responsible for establishing communication with the service according to the service communication protocol and API.

- **Composite service execution**

Composite services will be executed directly in the Palette Service Platform, by a dedicated engine, such as the Service Orchestration module if it is decided to include it. When a composite service is to be called, the user needs to make his requests according to the WADL or WSDL file that will be available to access this service.

The requester calls the Service Orchestration module with instructions to access a composite service. Then, the latter redirects the call to the dedicated engine, which interprets the message and executes the composition by calling the individual services. At last, it returns the corresponging data, or an error message.

- **Visual integration of selected services**

CoPs' members will have access to some functions or specific views of Palette Services through widgets that Palette partners will provide. These widgets will be accessible in the Portal. The user will be able to search for the widgets that she/he wants to see on her/his portal, which will be personalized to each user. Once a widget is chosen, the user can place it on the portal window. As soon as there are multiple widgets, the GUI enables users to compose widgets at their own convenience (e.g. spatial arrangement of widgets, etc.).

4.5 Description of a service

This section reproduces the global XML Schema agreed by the partners to formalize the description of a Palette service. This uniform and formal description is a first step towards interoperability. Each service must be associated with an XML instance file, valid with respect to the schema. The next figure shows a graphical representation of the schema. In the future, the schema could be changed or extended in order to support the description of extra information that can emerge from the provided Palette services. Currently, the XML instance files are stored on the Palette BSCW, but they will migrate to the PSRF as soon as it becomes available.

The schema serves several goals. First, it is intended to provide a standard description of each service so that services can be browsed, searched, and selected by the users through the Human Registry Interface. Such searches can thus be easily implemented with XML standards. The schema allows making multilingual documentation of services intended to a wide diversity of end-users. Furthermore, XML structures provided by the schema could later be annotated with RDF annotations in order to provide an additional level of search using semantic features.

The second - and main - goal of the schema is to provide a common description of services for machine access, so that services can communicate together in an automatic and consistent way. A service description may specify requirements that must be satisfied for calling and running the service correctly (for instance, a java runtime engine may be required on the local machine), as well as the address from which the application can be downloaded and installed on the local machine.

For describing Web services, additional XML languages are used in conjunction with XML Schema. Depending whether the Web service is based on the remote procedure call paradigm (e.g. using

SOAP) or on the Web resource paradigm (using REST), the XML instance file should respectively provide a link to a WSDL description of functions, or to a WADL description of exposed resources. This separate file may in turn refer to an additional XML schema that defines the type of the different replies.

For instance, a REST-based Web service x is completely described in an XML and machine-usable way by a triple (x.xml, x.wadl, x.xsd) where:

- x.xml is an XML instance file valid with respect to the schema proposed in this section;
- x.wadl is a WADL file that describes REST exposed resources as well as actions that can be performed on them;
- x.xsd is an auxiliary XML Schema that defines the type of the data sent in response to actions on resources.

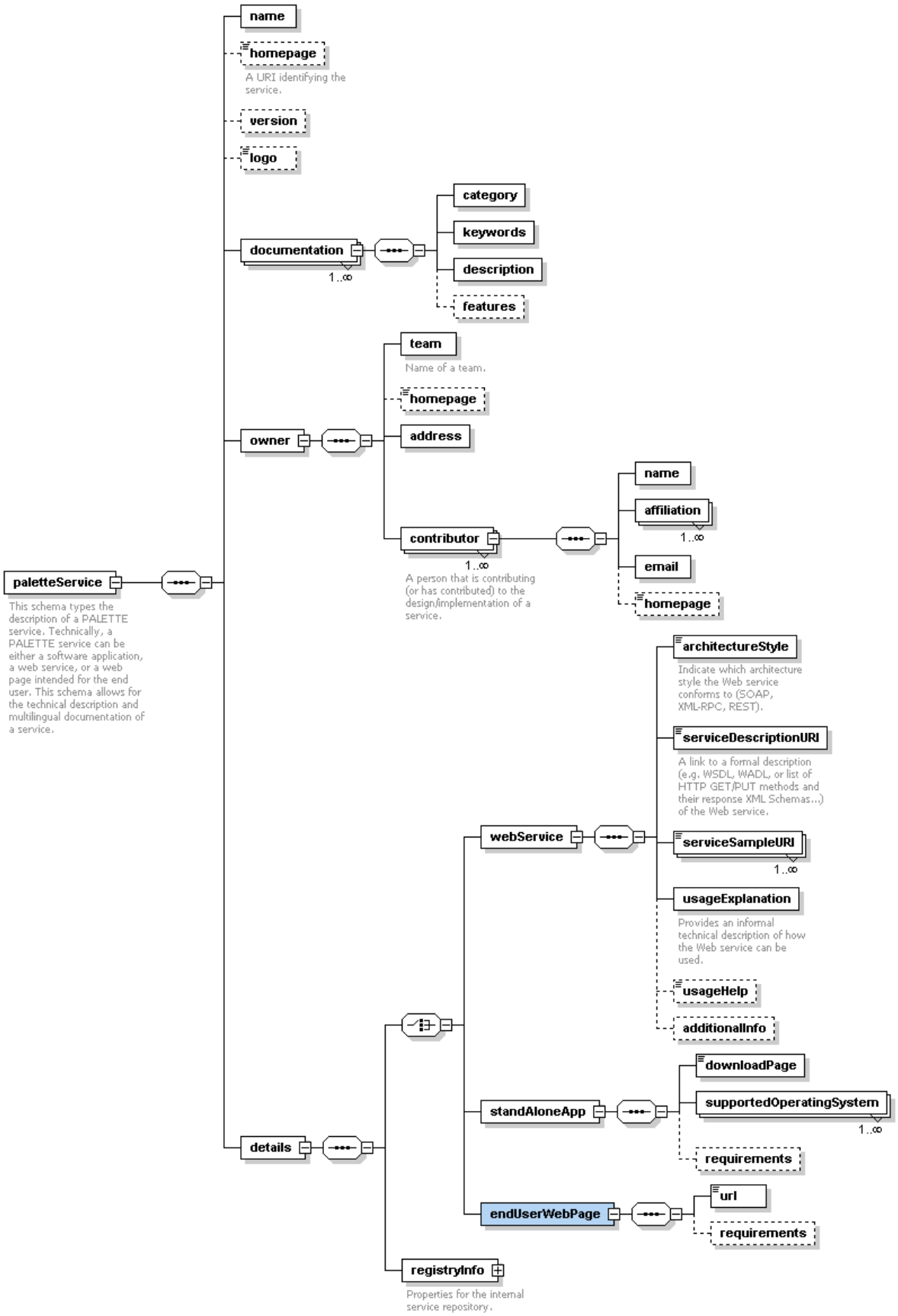

Figure 15. Graphical representation of the XML schema for description of services

4.6 Towards service composition with orchestration and choreography

The service orchestration module of the PSPA is proposed in prevision of future needs in terms of Web service composition. Globally, its functioning could be described as follows. First, we assume that there exists a language allowing to model new services as a composition of other services, and that these composite services can themselves be given a WADL or WSDL description, depending on whether they are accessible by the REST or SOAP way. Then, the service orchestration module's role is to answer to REST or SOAP calls on these composite services by executing the composition, which includes managing calls to atomic services and returning the results.

In order to find adequate solutions for the Palette context, we have performed a state of the art research on existing composition languages and frameworks. Today, there exist two main approaches to web service composition, namely, service orchestration and service choreography. In the following, we sketch some basic issues around them, the aim being to find the most suitable framework for our purposes in the Palette project.

Orchestration or choreography?

Classical definitions for orchestration and choreography are:

- **Service orchestration** is an executable business process that can interact with both internal and external Web services. It occurs at the message level and defines a long-lived, transactional, multi-step process model;
- **Service choreography** tracks the message sequences among multiple parties and sources. It manages the exchanges between services of different peers (partners).

To put it simply, orchestration deals with executable process, while choreography deals with multi-party collaboration.

In Palette, the composition of services should be seen from a global point of view, not from one's perspective. At the moment, service composition is seen as collaboration between two or more services needing to interact with each other. From this point of view, as already stated in D.IMP.03, services composition in Palette better matches with the choreography approach. However, if we take a CoP's point of view, "business processes" could be derived from the practices of a CoP, which could then be realized using an orchestration approach.

The following subsections present briefly the two current standards for orchestration and choreography, namely WS-BPEL and WS-CDL, respectively, and the frameworks that we have chosen for their suitability to answer future Palette needs in term of service composition. In particular, one of the choice criteria on which we focus is the handling of RESTful services, which has been agreed as the default communication mode between Palette services. More details might be given in a future D.IMP deliverable if Web service composition is finally used in the project.

Languages for service composition

WS-BPEL 2.0 (URL: WSB) represents business processes from a service view. It is compatible with WSDL 1.1, which is not designed for RESTful services. For the moment, there is no support for WSDL 2.0.

WS-CDL (URL: WSC) is a W3C standard, which describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behaviour. WS-CDL is compatible with WSDL 2.0 that is REST compliant.

Suggested frameworks

The selection of frameworks for Web service composition has been made according to the table below, which lists the characteristics we thought necessary for a good suitability to the Palette context.

Characteristic	Description
Open source	The framework has to be an open source solution

RESTful	It has to be compatible with RESTful services, but also SOAP services. It can be WSDL 2.0 compatible.
Choreography	Most of the frameworks are based on an orchestration approach (e.g. supporting WS-BPEL), but as choreography seems suitable to the current Palette context, the framework should support WS-CDL or another approach to choreography.
Execution engine	It needs to propose an execution engine, or to generate code compatible with an execution engine.

According to our study, we identified two suitable frameworks that will be further investigated: JOpera, which is orchestration-based, and Pi4SOA, which is choreography-based and supports WS-CDL. Some details are provided below.

JOpera

JOpera is proposed as an open source Eclipse plugin (URL: JOP). It is compatible with RESTful services. It can be used as a server that will provide Web service (WSDL) access to any deployed processes. Processes are OML (an XML language) files, which are created by the visual interface of JOpera that allows drawing the composition. The last version, 2.1.2 was released on January 11th, 2007.

Pi4SOA

Pi4SOA is proposed as an open source Eclipse plugin (URL: PIS). It leverages the new Choreography Description Language standard, developed by W3C, to provide tools that help build distributed Service Oriented Architecture (SOA) based systems through the principle concept of interaction. It allows service behaviour to be generated into Java code and then deployed in an Axis environment. Business logic can be added into the generated code and so the full solution can now be designed, tested and deployed into a distributed infrastructure.

Conclusions

Needs in term of advanced composition of Web services have for the moment not been reflected in scenarios proposed to CoPs. However, such compositions might arise from the ongoing research on defining more generic scenarios. Then, the orchestration module of the PSPA will help. The current Palette context leads to take the direction of a choreography approach to composition. However, as already quoted, an orchestration approach could also be useful. The two frameworks we have proposed will be further investigated mainly on the basis of what generic scenarios will provide in terms of services composition and also on the web services that will actually be registered into the PSPA.

4.7 Interoperability example: a cross-service document importer

Corresponding to the interoperability example given in subsection 2.4, this subsection describes - in the context of PSPA - two different ways to provide the cross-service document importer to CoP members. This will show how the PSPA can support either a data mashup or a SOA architecture style (a RESTful or a SOAPful implementation is possible in each case).

4.7.1 Common details to both architecture styles

In order to enable a cross-document importer Widget, a set of Palette tools has to provide – through a SOAP or a REST Web service (this is independent of the architecture style) – a document search function and a document retrieval function. The first function will either return a list of documents matching a given query, or an empty result if no match can be found. The second function will return a document from the service storage layer by taking as input the name of the document or a document handle such as a document id returned by the previous function, and the credentials of the requestor.

Although it is possible that these functions have different names and parameter names, it is simpler to suppose that they follow a common API definition. In the following, these two functions are called the "/export document/" Web service.

For each created "/export document/" Web service, the service providers have to create a service description complying with the specifications that are described in subsection 4.5. Then, service providers also have to register the "/export document/" service to the Palette Service Registry in order to make it available to other Palette partners. The procedure of the creation of service description together with the registration takes place through the PSRF Web-based Registry UI in a user-friendly way.

4.7.2 First solution without using a composition engine

Without a composition engine, it is possible to provide a solution with a data mashup architecture style that will require first that the user selects a service to query before retrieving a document. This is the procedure that was illustrated in subsection 2.4. This solution requires the following steps:

- first, an initialization step that consists of the invocation of a function to obtain the list of services with a document export capability in use by the CoP of the requesting user; this function must be invoked on a service that knows the CoP configuration;
- second, a document query step that consists of calling the search function in one of the exporting services with a document name as parameter;
- third, depending on the success or not of the previous step, a document retrieval step that consists of calling the document export function in the selected service.

The first step allows to populate a menu similar to the one presented in subsection 2.4 from which the user can select the service from which s/he wants to import a document. The second step is triggered when the user has input a document name and clicked on an upload button (for instance).

In a data mashup architecture style, all these steps are embedded on the client side in a cross-document importer Widget. Thus, the composition of service calls is implicitly hard-coded inside the Widget, for instance with Javascript code. The component Widget must also be initialized with the CoP name and the credentials for the user, for instance immediately after it has been loaded into the page. This can be done with a Javascript call too.

The cross-document importer Widget must be loaded from a mashup host server that will also route its service invocations to bypass the browser domain security policy. The mashup host server must also be the same that hosts the interactive service page inside which the cross-document importer Widget is embedded for the same reason.

The protocols used to invoke services can be either REST or SOAP. This depends on the implementation style of the implemented services, either RESTful if they are exported as resources, or SOAPful if they are exported as remote procedure calls.

Below is an imaginary example of a RESTful style resource invocation to retrieve a list of services with a document export capability, which corresponds to the first step of the above process:

```
GET /cops/adira/services.xml?can="document-export" HTTP/1.1
Host: www.palette.org
```

This supposes that a server is running at the "www.palette.org" network location. This server could be for instance an extension of the PSRF. It could also be a part of the CAKB. In a RESTful way, that server hosts the following resources :

- "cops", is a resource that describes all the registered CoPs ;
- "services", is a resource that describes all the services known to the PSPA.

The request is a GET request. According to the REST model, it will query a resource description from the target resource by invoking one of its index methods. The target resource is the "services" resource that manages the description of Palette services. The request will be interpreted in the following way:

- the "adira" part (it could also be a numerical database identifier of the given CoP) will be interpreted as a parameter of the "cops" resource that selects the Adira CoP within the cops; the resulting resource that describes the Adira CoP will be added as a parameter to the index method call on the target resource;
- the "can" part also adds a parameter to the index method call; in the above example, it contains a comma separated list of predefined capabilities that the service should implement in order to be listed in the answer;
- the "services" part routes the request to the target "services" resource.

The ".xml" suffix asks the "services" resource to send its response as an XML document. Taking into account its parameters, the index method of the "services" resource will then return a list of the "/export document/" services in use by the Adira CoP. The service invocation must in fact be routed through the mashup host from which the Widget has been loaded to allow cross-domain call. This is not described in details here.

The two other service invocations, for the query step and the retrieval step, would be of the same type, directly targeted at one of the "/export document" services returned on the first step. One level of complexity would be added to pass the credentials of the user and enable access to his/her private documents if required. Another level of complexity would also be added if each of the services uses a different API, as we have already discussed.

4.7.3 Second solution with a composition engine

By using a SOA architecture style, it becomes possible to benefit from the existence of service composition engine to create a "/cross-service document importer/" (CSDI) Web service[2]. This service will handle requests for documents, interpret them, and then forward them to all the PSRF registered "/export document/" Web services.

Using the CSDI to build the cross-service document importer Widget, it becomes possible to remove the step of the selection of a particular target service by the user and make a global search from a document name, providing a list of matching documents as a result (if it can be found in several services). That would slightly alter the mockup design presented in subsection 2.4 by removing the need for a service selection drop down list, and by adding a document result selection dialog step, which is not described in details here.

As with the previous solution, a document query to the CSDI could be encoded either with a SOAP or a REST protocol, depending on the type of implementation of the CSDI. Below is an imaginary example of SOAP remote procedure call for searching Document "PaletteMeeting5Nov07" created by any service:

```
POST /CSDI/CSDI.asmx HTTP/1.1
Host: www.palette.org
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.palette.org/CSDI/searchDocuments"

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

[2] It is also always possible to create a ad-hoc data mashup component that will "manually" make the composition of calls to all the available "/export document/" services it can discover; however, this is not envisaged.

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
   <AuthHeader xmlns="http://www.palette.org/CSDI">
     <Username>a2Fyb3Vzb3M=</Username>
     <password>a2Fyb3VzYW4=</password>
   </AuthHeader>
 </soap:Header>
 <soap:Body>
   <searchDocuments xmlns="http://www.palette.org/CSDI">
     <tool>*</tool>
     <document_name>PaletteMeeting5Nov07</document_name>
   </searchDocuments>
 </soap:Body>
</soap:Envelope>
```

This supposes that the CSDI service is running at the "www.palette.org" network location. This also supposes that a wildcard character inside the "tool" parameter instructs the CSDI to search in all its registered search services. The CSDI would most probably interact with the PSRF to get the list of the "/export-document/" registered services.

When interpreting a document query, and when the CSDI gets a document as a response from a particular Web service, it adds it to its result list. At the end, it returns the list of documents back to the original requestor in an XML file or and XML error file. The retrieval of the document selected by the user from the result list is not described in detail here; depending on the implementation choices, it could be directly invoked on the document owner service, if returned in the result list, or it could also be mediated through a retrieval function of the CSDI.

In the creation phase of this global new CSDI service, developers have to perform some search queries to the PSRF by the Web-based Registry UI aiming at finding a list of the appropriate Web services that support the functionality "/export document/". For each available service, the developers examine the Request/Response methods and parameters and map the service response messages to a unique set of messages that will be returned back regardless the selected service. In case where the "/export document/" service require the credentials of the requestors, the global service will be responsible for taking them from the client and passing to the services.

To avoid specific developments that are required by this composition example, developers can use a service composition framework providing a visual interface like Pi4SOA and JOpera, which are investigated in the context of WP5 and referenced in subsection 4.6. In that case, the CSDI composite service would be defined as a composition of services in a scripting language like e.g. BPEL and the dedicated composition engine of the PSPA would execute dynamically the composition (be it through orchestration or choreography) in response to SOAP or REST calls.

When the CSDI service is ready, the developers have to register it to the Service Registry by using the same Registry UI. A service description for this composite global service is also required for this step.

When the CSDI service is available in the Palette context, the Palette tools developers will be able to find it through the Service Registry. They can then decide to embed calls to the CSDI Web service from one of their interactive service, either from within a custom extension of the service UI, or by reusing an hypothetical cross-document importer Widget that would be available in a UI component library. An alternative solution is to create a CSDI Widget for the Palette Portal. This would allow to query document from outside a given interactive service. Note that the creation of a Portal widget is also possible with the first solution; in that case, the Portal application would behave as a mashup host.

5 Guidelines for Homogeneous User Interfaces

This section focuses on the interactive services. It addresses the issue of UI integration, which is a main concern of Palette, as members of CoPs will most probably interact with several different interactive services (in parallel or in sequence) to perform actions.

5.1 Rationale

The goal of these guidelines is to propose some standards and some practices that, applied together, would lead the users of Palette services to perceive them as an integrated working environment, and not as a disparate set of applications. These guidelines can be read as a reference set of standards and practices that support the application of the ergonomic guidelines and usability analysis which are conducted in WP1. Both efforts aim at fulfilling Palette goal of usability and acceptability.

The work done in WP1 focuses on the individual services' UI and on the users' needs, whereas in WP5 we try to answer these requirements from the point of view of UI integration at the developers' level. This section has several goals, as it aims at promoting the visual identity of the Palette project, maintaining the usability of the integrated environment and suggesting tools and methods for interface consistency. Thus, this work could lead to the identification and centralization of graphical resources, and, maybe, to the creation of a library of Palette UI components, in order to share efforts and to simplify maintenance issues.

This section addresses more specifically interactive services which are available as Web applications or as Web UI components and not as standalone desktop applications. The reason is that standalone applications are more specifically tied to their respective native runtime platform(s) and should follow their corresponding usability guidelines. However, they could also be considered in the future.

5.2 User Interface composition styles

The creation of composite applications from reusable components has been well studied in software engineering and data management. However, it received less attention at the presentation level. User Interface integration is becoming an active research issue (Daniel et al., 2007).

The easiest case of composition is to extend an existing interactive service with a well-defined UI component that usually fits within a graphical box and that can be rendered by the graphical engine of the runtime environment. In that case, the new UI component can be embedded into the layout of the interactive service. Depending of the type of functionalities it adds, the extension may also require an inter-component communication mechanism. To some extent, this is also the composition technique which is employed to extend a browser with a plug-in that gives access to new functionalities. A similar concept, to extend a particular Web application with a plug-in, would be very useful. However, there are still no universal solutions due to the lack of standardisation of Web applications presentation layers above the level of XHTML.

More complex cases happen if the purpose is to aggregate several Web services to build a new interactive service. The most common solutions in that case are the Web portal, and the Web mashup.

The Web portal distinguishes the UI components, or Widgets, from the composite application, or Portal, from where they are executed. By following conventional APIs, the Widgets can be graphically composed in a single page, thus creating customized composite applications.

The Web mashups are created by associating components that may come from different origins. These components retrieve data from different sources, through a mashup host from where they are executed, and they may be rendered inside a browser to show a representation of the data. Some components offer an API that can allow to manipulate the data and to change their appearance. Some very popular components in cartography for instance, allow displaying layered data coming from different sources on top of graphical maps.

The UI composition styles is a topic that should be further investigated, as the interoperability needs will require to compose either existing interactive services with new Web services, or to aggregate

Web services to create new interactive services. It seems essential to identify the solutions that will require the least effort to stay compatible with the already existing interactive services without imposing them a complete UI redesign for interoperability reasons.

5.3 Inventory of common Palette UI elements

The common UI elements correspond to high-level widgets, or more basic interactive graphical elements, which are present in the different interactive services. The following table shows a list of common UI elements, as of October 2007. This work is still in progress (some interactive services have not been integrated into the list for the moment).

Table 8. First inventory of common UI elements.

	Bayfac	**Cope-IT**	**e-Logbook**	**Generis**	**Portal**	**SweetWiki**
Palette Logo			x		x	x
Tool Name	x	x	x	x		x
Search Box	x	x	x			x
Login		x	x	x	x	x
Logout		x	x	x	x	
Sign In		x	x	x		x
Link to Help		x		x		x
Feedback Button		x				
File Upload	x	x	x		x	
WYSIWYG Text Editor			x	x		x
Menu bars	x	x		x		x
Tabs		x				x
Contextual menu		x				x
Accordion[3]		x				x
Calendar			x			
Star Rating			x			
Tree				x		x
Toggle box	x					

It is interesting to note in the table above that some of the most common widgets used in Web sites do not appear in Palette tools and services or are very rare such as the basket, the comparator or the feedback button. It is not that these widgets may be necessary, but it may be interesting to understand why they are not needed and eventually to reconsider to use them in design especially if they are familiar to the users. For instance, a feedback button that sends an email to a well-defined authority could easily be integrated into every Palette Web application, as it is already the case in CoPe_it!.

[3] The accordion menu is a two-levels menu that allows users to first select a category and then select from items within that particular category; the apparition of the items from the new category that replaces a former one is animated; the accordion menu usually takes a fixed screen space.

As we can see in the table above, several UI elements are present in more than one service and then could be harmonized. Some cross-usability analysis between the common UI elements would reveal some inconsistencies such as different labelling for the same functionality (e.g. 'help' vs. 'documentation', or 'sign in' vs. 'register'). The following section makes some suggestions to harmonize the look and feel.

5.4 Look and feel

5.4.1 Static graphical characteristics

In a classic Web application, the user interface is presented to the user as a set of pages. The static graphical characteristics of each page are the characteristics that do not imply any interaction with the user,.

The following recommendations address the main static graphical characteristics of a Web GUI in the context of the Palette project. Most of them can also be considered for application to standalone desktop applications.

Dimensions

The minimal screen size to take into account is 800x600. So, all Web interfaces should be usable with this resolution (URL: WSG).

The pages of a classic Web application define an overall navigation structure which can be characterized by a maximal depth and a maximum breadth. Typically the maximum depth also corresponds to the maximum number of clicks required to reach a target in the application. The maximum breadth corresponds to maximum number of clickable links on a page. A lot of usability research has been done about the optimal breadth/depth ratios, either in menu structures or in inter-pages navigation structures. The general conclusion is that breadth is better than depth (Larson & Czerwinski, 1998). This has lead for instance to the recommendation to use a navigation bar on the top, or a vertical navigation menu on the side, of each Web site. Palette Web interfaces should avoid to be spread into deep page hierarchies. This is particularly important from the integration point of view since a typical user of Palette services may interact with several services in parallel or in sequence to perform a single task, thus increasing the perceived "depth" of the integrated service.

Text and icons

The Palette logo is the cornerstone of the visual identity of the project; it must be present on all Palette services. A reference file of this logo is available in the "Palette Logo" section of the BSCW[4].

Figure 16. Palette logo.

The graphic style of each application is up to each application designer, but as default s/he can borrow the elements that come from the graphic charter of the Palette main Website. For example the color palette of this Website is basically constituted of a gradient of grey slightly blue (from #f4f4f4 to #cecece), blue (#036) and red (#c00). Some extra colors are available in the logo, so as to illustrate the concept of "palette" and could be used in the interface to identify some parts of the Palette, these colors are yellow (#fc0), red (#c00), green (#090), dark blue (#036).

[4] https://bscw.ercim.org/bscw/bscw.cgi/146107

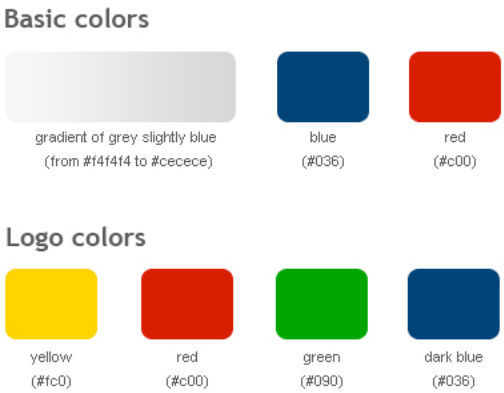

Figure 17. Suggested color palette.

Regarding typography, it seems better to use a medium-sized sans serif font for screen legibility[5]. By default, and according to the graphic charter of the Palette main Web site, the following fonts can be used in this order, depending of their availability: Verdana, Arial, Sans Serif. The two first one come from the Microsoft core fonts for the Web (URL: CFF) that are available on all current operating systems (Windows, Mac OS X, Linux, ...), and the last one is a fallback to notify the system to use a Sans Serif font (URL: OPQ170 and OPQ73). The color of this font must be black or white, depending of the background color, in order to maximize contrast. One can find additional guidelines on Web typography in a Nielsen's Alert box (URL: ALU).

Each Icon is a small graphical illustration of a concept. When dealing with several applications, the risk is that there are several representations for a same concept. Thus, it seems better for memorization to show to the user always the same icon for the same concept, and by extension it seems better to use a common icon set across applications. An example of open source icon set is the one used by the gnome desktop on Linux, the tango project (URL: TTP).

Common UI elements

Common UI elements must - as much as possible - be identical across applications.

We can categorize the common UI elements in three groups:

- **mandatory:** this category contains all UI elements that enables the user to recognize the application.
  - Palette logo
  - interactive service name
- **recommended:** some UI elements are recommended as they improve the global ergonomics of the application.
  - search box, as a mean to access full text
  - link to Help
  - navigation menu
- **optional:** all the other UI elements.

All these elements may not be realistically homogenized in full, however the harmonization process seems really important for three types of UI elements:

- **navigation:** in a Web application, navigation can often become tricky, that is why most of them use navigation menus, represented as a list or a tree;
- **help:** users may not often use help, however it is used either by the beginners, and as a last resort by all users. Thus, help should be very easy to reach. The links to help pages can be general, for

[5] some experts tend to agree with that like in (URL: TBF), but other studies shows that there is no clear dichotomy (URL: WAR)

example in the main menu, or contextual, with links corresponding to parts of the current view. In this last case, they must be easily recognized, with an icon for example;

- **complex widgets:** some widgets are an aggregation of other more simple widgets, and leverage the cognitive load on the user. The most common example is perhaps the WISYWIG editor (or rich text editor), which embeds a simplified HTML editor in a widget. In an ideal world, these widgets should be the same on all Palette services, in order to bypass the learning curve encountered each time a user has to work with another variant of the same complex widget. Several open source implementations are available for this kind of component such as FCKeditor (URL: FCK), TinyMCE (URL: TMC), and Kupu (URL: KUP).

At a global level on the Palette project, the coherency on the three main categories of UI elements can be homogenized, in priority over the other types of UI elements. The next related deliverable could provide a Palette "standard" representation for common UI elements as a reference for all current and future graphical developments. But this fact will probably depend on the availability of a graphical designer in the project.

Layout

The page layout can be described as a set of guidelines about the position of recurrent elements in a Web page. Due to the diversity of Palette tools, it is not possible and desirable to recommend a single layout for all of them. However, it is possible to propose layout guidelines for common UI elements. For example, the following list gives the frequent position in a page of some common UI elements on many Web sites:

- tool name: top left
- search box: top right
- form validation buttons: bottom left
- navigation menu (very often primary or secondary navigation system): top left
- logout: top right
- link to help: top right
- tab group (very often primary or second navigation system): top

5.4.2 Dynamic graphical characteristics

The dynamic graphical characteristics of an application are the characteristics of the display that change with user actions (e.g. feedback) or with time (e.g. animations). The dynamic graphical characteristics contribute to the perceived atmosphere, or ambiance of an application, as importantly as the static characteristics. They contribute directly to the usability by helping the users to perceive the changes of state of the application. They also contribute indirectly to induce emotions, which have been recently acknowledged as contributing to the usability of an application (Norman, 2004). That trend is at the heart of the recent development of the so-called rich user interfaces in the Rich Internet Applications (RIA) and Rich Desktop Applications (RDA).

This section starts first with a general guideline concerning the graphical behaviour that Palette applications should adopt when the user resizes the browser window or when s/he scales the page. Then it defines some of the advanced effects that are at the base of RIA. The purpose here is to give some directions in which Palette UI could evolve to take benefit from recent advances in Web UI frameworks in order to increase usability and to develop a common user experience.

Resizing and scaling behaviour

The resizing behaviour is triggered every time the user resizes the browser window. Depending on the structure of the page content, the choice of units for measures, and the style sheets, the results may be quite different. For instance text areas may grow and the text may fill these areas (also known as "liquid layout" or "fluid layout"). On the contrary, all the page content may remain identical, the only changes being the apparition of empty spaces at the borders or of scroll bars if the window become too

narrow to display the full content (also known as “fixed layout”). The elastic design (URL: ED) alternative is a good alternative for Palette as it is a good compromise between screen space optimization and legibility constraints. In that alternative, the layout itself should also expand or collapse depending on the window size. The text should remain the same, only spaces between text and borders being expanded or collapsed. For user interfaces that cannot fit into one page, the expansion/collapse behaviour applies only to the horizontal dimension.

The scaling behaviour is triggered every time the user changes the font sizes with a menu option usually called “smaller” or “bigger”. The article cited above also gives some clues about properly styling text and layout elements so that they scale correctly on all browsers. From an integration point of view it would be preferable that all services exhibit a similar scaling behaviour in all the browsers.

Advanced effects

RIA can be recognized through the use of advanced graphical effects and animations. Combined together those allow bringing inside the browser interaction techniques that were only available up to now in desktop applications or in video games. It is difficult yet to predict exactly the benefits of RIA over other more classical user interfaces; however it is possible to foresee:

- a better learnability, especially if a graphic designer has worked to augment the expressivity of the graphics;
- a better legibility because the visual density is alleviated, for instance by putting data into superposed layers which are dynamically displayed only on-demand;
- a better output bandwidth as more information can be contained in one page, for instance when parts of the page can be contracted or dilated to reveal more information (e.g. accordion menu) or when Ajax techniques are used to refresh only parts of a page when data and/or application states are modified; this way a single page can be used to display more data than a static one using the same screen real estate;
- a better input bandwidth because point-and-click can be complemented with new interaction techniques such as drag-and-drop or temporal-based selection in animated menus.

For all these reasons, Palette service developers should explore design solutions based on advanced effects. Of course, a drawback to using new interaction techniques is that there may be a new learning curve imposed on users. For that reason too, it seems reasonable to require that Palette services start first with a small number of advanced effects and be coherent in their choice. An interesting side effect of that strategy could be also to generate the feeling of a uniform Palette user experience. That would help the user to see all the interactive services as an integrated environment.

The following list describes the advanced effects that could be experimented first, based on their expected benefits:

- displaying information in layers visible temporary whenever it is possible to make a hierarchy of information contained in a page (e.g. generalization of tool tips, to give secondary information about document authors, creation dates, tags, etc. instead of cluttering the screen with these data);
- animating local updates of a page if Ajax is used to incrementally update the screen; a common way to signal such changes is to “flash” a yellow rectangle over the area of change before displaying a new visual state;
- systematically displaying an animation (e.g. a spinning wheel) when the user has to wait;
- drag-and-drop to apply commands on objects when the objects and commands can be represented as graphical objects on the screen.

The last advice aims at introducing direct manipulation whenever possible (Shneiderman, 1983). An interesting source of inspiration can be found in the Yahoo UI Design Pattern Web Site (URL: YDP).

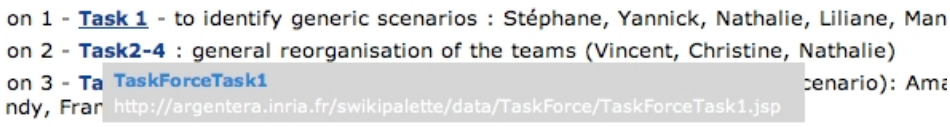

Figure 18. Example of a dynamical layer effect to display extra information when flying over a link in SweetWiki.

One possible consequence of using advanced effects may be to lower the accessibility of a site, by hiding some information or some application states into data structures which are not available to the accessibility software and devices. Another consequence is to require recent browser supporting the latest variants of scripting languages and Web API. These risks can be reduced by a programming approach which is known as "unobtrusive Javascript" (of course when Javascript is used as the programming language of choice). That approach requires one to separate all the programming code from the main page from which the UI is loaded, and add all the dynamical advanced effects only after the page is first displayed by the browser (i.e. on the "load" event). By doing so, if the scripting language is not enabled, at least the basic will be displayed. Of course, the basic page should enable a minimum of end-user interactions to make the service operable.

5.5 Usability, accessibility and internationalization

The integration of the services should not lower the usability of each individual service. In particular the usability recommendations and analysis of WP1 is the principal source of information that should drive the design of the UI of interactive services. In particular the document "transversal ergonomic guidelines" available on BSCW should be consulted by all the implementers. This is true also of the individual Palette service evaluations which are already available. As explained in the transversal ergonomic guidelines, the classical usability guidelines to apply can be found in (Bastien et al., 1993; 1998), (Nielsen 1990) and (Nogier, 2005).

For accessibility issues, service providers can refer to the accessibility principles and design ideas of W3C, as expressed in (URL: WCA).

In the international context of Palette, a weak form of internationalization could be achieved by imposing one common language and one common writing system. However, some CoPs have explicitly requested a localized version of the tools for their mother tongue. Thus, we recommend that Palette tools should be internationalized by supporting the local languages and writing systems of the end users. In particular. this imposes a strong constraint on the character encodings that should be supported by all the tools that can input text characters, a minimum requirement for French speaking CoPs that expressed the needs for localization seems to support at least UTF-8 in text inputs and text displays.

5.6 Technical recommendations for applying the guidelines

This subsection addresses the benefits that could be gained from common implementation languages for UI. Then, it defines three conformance levels that can be used to measure the level of homogenous UI integration achieved by an interactive service. Its goal is to orient the efforts of development teams. The last part is very practical: it lists some tools that can be used during development to test and validate the application of the guidelines.

5.6.1 Implementation languages

These guidelines for homogeneous UI integration do not impose any implementation language or client-side UI architecture. This may evolve in the future, for instance to allow a better integration by allowing some UI code sharing between interactive services. At the limit, if some interoperability needs can be synthesized into UI widgets, it would be possible to provide a common library of Palette UI components for inclusion in interactive services.

In the meantime, Palette Web UI should be implemented with open standards, such as valid HTML or XHTML and CSS, with a preference for the latest versions of the W3C standard which is at the Recommendation Level, in order to not compromise a possible factorization in future versions of the platform. For advanced effects that require new capabilities, the canvas element or SVG can be considered as alternatives to the Flash proprietary format.

Interactive services should be targeted at least to execute in the following browsers: Firefox 2, IE 6/7 and Safari 2. However, it would be interesting to make a survey of the browsers in use by CoPs members to better inform that suggestion.

Currently, advanced effects can be programmed with off-the-shelf libraries. As many of them are available, and the number is still growing, it is difficult to recommend one over the others. If some Palette services adventure on the ground of RIA, it is sound to go further into benchmarking and evaluating some libraries in order to incorporate them into future versions of the implementation guidelines. The benefits for integration of suggesting a limited set of preferred RIA libraries would be to homogenise the performances and the look and feel of the applications, as most of the libraries come with predefined high-level effects.

5.6.2 Conformance levels

In order to validate the guidelines that are presented above, we propose a checklist of the main points, classified in three levels of conformance, ordered by complexity.

Basic conformance level:

- the interface should work with minimal screen size of 800x600;
- the Palette logo should be present in the application;
- the application should work with the following list of browsers: IE 6/7, Firefox 2, Safari 2;
- mandatory common UI elements should be present;
- common UI elements should be positioned according to the above recommendations.

Intermediate conformance level:

This level includes all criteria from the basic conformance level.

- the Palette graphic charter and typography should be respected;
- the layout of the page should be elastic;
- XHTML and CSS codes should be valid;
- the UI should respect WCAG 1.0;
- the UI should respect basic usability principles.

Full conformance level:

This level includes all criteria from the two previous conformance levels. Some of these points are not precisely defined, and probably require a collaborative definition; this is a work in progress.

- the UI should be internationalized in the main languages of the users;
- the UI should use the common icon set;
- the UI should use homogenized elements for navigation, help and complex widgets;
- if the UI use advanced effects, it should respect recommendations regarding these effects.

5.6.3 Test and validation

The following resources and tools can be used to test a UI, notably with the conformance process:

- screen size: the Web developer toolbar for Firefox enables to test a Web site at any screen size https://addons.mozilla.org/en-US/firefox/addon/60

- code validation: HTML and CSS source code can be validated through W3C validators (validator.w3.org/ and jigsaw.w3.org/css-validator/) another possibility is to use HTML tidy Firefox extension (users.skynet.be/mgueury/mozilla/)
- accessibility: it is possible to check the colors of an interface against color-blind disability with vischeck (www.vischeck.com/vischeck/vischeckURL.php)
- accessibility: the conformance to several accessibility standards (including WCAG 1.0) can be checked with ocawa (www.ocawa.com/en/Test-your-Web-Site.htm)
- browser support: this can be evaluated by using browsershots (browsershots.org)
- best practices: opquast is a list of best practices for the quality of on-line services (en.opquast.com)

6 Conclusion

The integration of services in Palette can advance one step further by following the guidelines for development proposed in this deliverable. The proposed PSP architecture introduces some modules, such as the PSRF, that are essential to share a mutual awareness of the available services specifications. This allows going a step further than a simple folder on BSCW containing a set of XML files. Moreover, with the development of presentation modules such as a composable Portal and a Registry UI, it will be possible to access the PSRF and Palette services through a unified entry point. This will allow developers of services and CoPs members to get a place from where they can start their Palette user experience.

The PSP can be extended towards service orchestration, if this is required in scenarios. According to the current state of interactions between Palette Services, the Palette context still raises interoperability issues. Interactivity between Palette services is addressed case by case and this situation creates tight coupling, thus leading to a low level of global interoperability.

However, one important step to higher interoperability has been made: a common communication mode for services interactions, with the adoption of a standard for the syntax of exchanged messages. Palette services are now described according to the Palette Service Description Model and offered Web services are detailed with a preference to use WADL, providing a REST interface. HTTP has been chosen for communication and RESTful Web services are the vehicles for exchange of XML data. According to an interoperability maturity model such as LCIM (Levels of Conceptual Interoperability Model) (Turnitsa, 2005), we have the entire technical infrastructure to get a level 2 on a [0-5] scale, namely syntactic interoperability.

The scenarios currently do not make use of direct service-to-service calls at the integration level, except for the e-Logbook - CoPe_it! case. Indeed, most of the interoperability needs concern data integration (i.e. data or metadata sharing). If better technical interoperability were sought, the next step would be for each service to provide more functions accessible through REST. Moreover, interaction of Palette Services with already existing CoPs tools will have to be explicitly considered. If interoperability is sought for the whole CoP environment, the existing tools must be adapted to comply with the Palette Services that will need to have access to data and metadata handled by these tools and be aware of what happens in them.

The final step to reach the LCIM level 3, which is semantic interoperability, would be then to define a common vocabulary for (meta-)data exchanges between services. For that, the ontologies developed in WP3 or extensions of these, might be useful. In particular, this would allow aligning the meaning of concepts used by different services to common ones (defined in ontologies which correspond to general definition of what a CoP is and how it works). Such use of a common model would finally help to avoid making local adaptations on a service-by-service basis.

Finally, this version of guidelines has also raised and elaborated issues related to the consequence of Palette integration on the usability of the interactive services. Some first propositions targeted at developers have been done to maintain the consistency of the user experience. From a technical point of view, it also opens interesting perspectives on sharing some UI components, or Widgets, that could be embedded and reused inside of interactive services to get a common look and feel. In particular, that would be an achievement if some interoperability needs could be covered by such components.

The next steps are to develop the PSP and the corresponding presentation modules. Then, using these modules, the PSRF must be populated with services descriptions and their REST exposed resources, as well as actions that can be performed on them. That work must be done in parallel with the refinement of the scenarios in Task 4 of WP5. The match between scenarios interoperability needs and Web service functions must be continued. In some cases, interoperability needs will certainly require to find innovative solutions, including the introduction of new services into existing tools, the adaptation of existing interactive service UI or the development of new Widgets, to be included in the portal, in existing interactive services or in new mashup UI. This should progressively lead, following the participatory design methodology, to the emergence of a Palette composite environment accessible through a portal.

7 References

7.1 Bibliography

Bastien, J.M.C. and Scapin, D.L. (1993). Critères ergonomiques pour l'évaluation d'interfaces utilisateurs. Rapport technique INRIA n° 156, Juin 1993, INRIA : Le Chesnay.

Bastien, J.M.C., Leulier, M. and Scapin, D.L. (1998). L'ergonomie des sites Web. In Créer et maintenir un service Web, cours INRIA 28 Sept - 2 Oct 1998, Pau (France), ADBS Editions : Paris.

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.

Dontcheva, M., Drucker, S.M., Salesin, D. and Cohen, M.F. (2007). Relations, Cards, and Search Templates: User-Guided Web Data Integration and Layout. In Proceedings of UIST'07, October 7–10, 2007, Newport, Rhode Island, USA, ACM, p. 61-70.

Fielding, R.T. (2000). Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000.

Erenkrantz, J. R., Gorlick, M., Suryanarayana, G., and Taylor, R. N. (2007). From representations to computations: the evolution of Web architectures. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ESEC-FSE '07. ACM Press, New York, NY, p. 255-264.

Halevy, A. Y., Rajaraman, A. and Ordille, J. J. (2006). Data integration: The teenage years. In Proc. of VLDB, p. 9–16, 2006.

Larson, K., and Czerwinski, M. (1998). Web page design: Implications of memory, structure and scent for information retrieval. Proceedings of CHI 98, New York, NY: ACM, p. 25-32.

Liu, Xuanzhe, Hui, Yi, Sun, Wei and Liang, Haiqi (2007). Towards service composition based on mashup. 2007 IEEE Congress on Services, 9-13 July 2007, pp. 332-339.

Marache K. & al. (2007). Palette, Draft Implementation Plan 2 (M13 – M30), Period 01/02/2007, Version 3.0, 31/07/2008, available on request on private Palette BSCW.

Naudet Y. (2007). A proposition for a general Palette Services Platform, WP5 technical meeting, Luxembourg, 8 Feb. 2007, slides available on request on private Palette BSCW.

Nielsen, J. (1990). Ten Usability Heuristics. Online article available (as of October 2007) available online at www.useit.com/papers/heuristic/heuristic_list.html

Nogier, J.-F. (2005). Ergonomie du logiciel et design WEB. Le manuel des interfaces utilisateur. Dunod, 3e édition, 272 pages.

Norman, D. A. (2004). Emotional Design: Why We Love (Or Hate) Everyday Things. Basic Books.

Peltz, C. (2003). Web Services Orchestration and Choreography, Computer, IEEE, Oct. 2003.

Shneiderman, B. (1983). Direct Manipulation: A step beyond programming languages, IEEE Computer 16(8), August 1983, p. 57-69.

Snell J. (2004). Resource-oriented vs. activity-oriented Web services, A quick look at the relationship of REST-style and SOAP-style Web services. IBM developerWorks Web site, 12 Oct 2004, available online at www-128.ibm.com/developerworks/webservices/library/ws-restvsoap/

Turnitsa, C.D. (2005). Extending the Levels of Conceptual Interoperability Model. Proceedings IEEE Summer Computer Simulation Conference, IEEE CS Press.

7.2 Webography

All the Web references have been accessed in October 2007.

ALU: Alertbox: Let Users Control Font Size, Jakub Nielsen, 2002-08-19, at www.useit.com/alertbox/20020819.html

CAS: Windows CardSpace at cardspace.netfx3.com/

CFF: Core Fonts for the Web at en.wikipedia.org/wiki/Core_fonts_for_the_Web

ELD: Elastic Design at www.alistapart.com/articles/elastic/

FCK: FCK Editor at www.fckeditor.net

GME: Google Mashup Editor at code.google.com/gme/

WSG: Web Style Guide, Lynch and Horton, 2004 at Webstyleguide.com/page/dimensions.html

JOP: JOpera at www.jopera.ethz.ch/

KUP: Kupu Editor at kupu.oscom.org

LIA: The Liberty Alliance Project at www.projectliberty.org/

MPF: Microsoft Popfly at www.popfly.ms/

MUL: Mule open source ESB (Enterprise Service Bus) and integration platform at mule.mulesource.org/display/MULE/Home

OPI: OpenID at openid.net/

OPQ170: Opquast, Best Practice N°170, A generic font family is given as the last substitution element at en.opquast.com/bonnes-pratiques/fiche/170

OPQ73: Opquast, Best Practice N°73, The number of fonts used throughout the site is limited to three at en.opquast.com/bonnes-pratiques/fiche/73

PIS: PiSOA at www.pi4tech.org/tiki-index.php

TBF: The best faces for the screen, Daniel Will-Harris, 2002 at www.will-harris.com/typoscrn.htm

TMC: TinyMCE Editor at tinymce.moxiecode.com

TSA: The Silo approach at www-128.ibm.com/developerworks/webservices/library/ws-antipatterns/

TTP: The Tango project at tango.freedesktop.org/Tango_Desktop_Project

WAR: Which are more Legible, Serif or Sans Serif Typefaces? at www.alexpoole.info/academic/literaturereview.html

WCA: Web Content Accessibility Guidelines 1.0 at www.w3.org/TR/WAI-WEBCONTENT/

WSB: Business Process Execution Language 2.0 at www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

WSC: Web Services Choreography Description Language Version 1.0 at www.w3.org/TR/ws-cdl-10/

YDP: Yahoo Design Patterns Library at developer.yahoo.com/ypatterns/

YPI: Yahoo Pipes at pipes.yahoo.com/pipes/