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.05 – First version of PALETTE Integration: Conceptual and Technical Integration

Due date of deliverable: 31 January 2008
Actual submission date: 12 March 2008

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

Organisation name of lead contractor for this deliverable: UNIFR and CTI

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

Keyword List: Generic Scenario, Activity Theory, Participatory Design, PALETTE Services, Support Services, Value Chain, Information Integration, Communication Integration, Platform Integration, Presentation Integration, Global Search, Single Sign-On, Common Resource Repository, Notification of Events

Responsible Partner: UNIFR and CTI

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
0.1-0.9	Up to 15-01-08	Draft	All
1.0	16-01-08	Draft	Liliane Esnault
1.1	22-01-08	Draft	Yannick Naudet
1.2	25-01-08	Draft	Nathalie Deschryver
1.3	2-02-08	Draft	Liliane Esnault
1.4	4-02-08	Draft	Aida Boukottaya, Liliane Esnault
1.5	11-02-08	Draft	Nikos Karousos
2.0	11-02-08	Draft	All
2.1	17-02-08	Draft	Liliane Esnault
2.2	20-02-08	Draft	Nikos Karousos
3.0	22-02-08	Draft	Nikos Karousos (version sent to evaluators)
4.0	29-02-08	Draft	Liliane Esnault and Nikos Karousos (version sent to the SC)
5.0	12-03-08	Final	Nikos Karousos (version sent to SCO and AFC)

Deliverable manager

- Liliane Esnault, EM LYON (conceptual integration part)
- Nikos Karousos, CTI (technical integration part)

List of Contributors

- Liliane Esnault, EM LYON
- Yannick Naudet, CRP-HT,
- Nathalie Deschryver, UNIFR
- Aida Boukottaya, UNIFR
- Nikos Karousos, CTI
- Manolis Tzagarakis, CTI
- George Gkotsis, CTI
- Nikos Karacapilidis, CTI
- Marie-Laure Watrinet, CRP-HT
- Alain Vagner, CRP-HT
- Stephane Sire, EPFL
- Michael Chiu Man Yu, EPFL

List of Evaluators

- Denis Gillet, EPFL
- Bernadette Charlier, UNIFR

Summary

The purpose of this deliverable is to provide a first version of PALETTE integration, distinguishing between conceptual integration and technical integration. More specifically, the conceptual integration concerns the design, implementation and validation of generic scenarios. Generic scenarios are built from the previously developed specific scenarios, insist upon the features which are common to different CoPs, provide evidence for the interactions that take place between all actors (human and non-human) for fulfilling CoPs activities, and show the intricacy of different functions of services that are necessary to achieve these composite activities. Generic scenarios also suggest possible improvements to better sustain the CoPs' development by innovating both at organisational and technical level (not just help doing what is currently done, but find new ways of doing things or new things that are made possible). Generic scenarios enable the implementation of such improvements through innovative architectures and interfaces, which take into account the possibilities of technological novelties made available from PALETTE services, the new organisational potentialities opened by the work in CoPs, and the learning capabilities of people involved in them. As recommended in the first year's project review report, the focus is made on integration issues. The concept of generic scenario is the way to precisely and thoroughly describe how the different functions of PALETTE services will have to interact and how they will be integrated together with users' actions in order to achieve the goals of sustaining, developing and enhancing the practices of the various PALETTE CoPs.

The technical integration concerns the adoption of relevant standards and development guidelines that facilitate interoperability of services, as well as the implementation of appropriate software components for integration and delivery of PALETTE services. Four dimensions of integration are addressed, namely the information, the communication, the platform and the presentation integration. PALETTE developers aim to both propose and apply solutions in order to satisfy the needs associated to each of these dimensions. Technical integration comes to support the conceptual one, by designing, implementing and finally, providing ways to allow the usage of integrated functionality by PALETTE CoPs. Thus, CoP members will have the ability to easily use one or more PALETTE tools or services, combine functionality of different services, and perform fundamental actions (such as store, retrieve, search etc.) on the full range of PALETTE tools and services according to the Generic Scenarios.

Table of Contents

1 Introduction

The purpose of this deliverable is to provide a first version of PALETTE integration. We distinguish between conceptual integration and technical integration. Conceptual integration answers the question: 'how to conceptually integrate PALETTE services together with CoPs practices and environments, as expressed in scenarios?'. For this purpose, the specific scenarios presented in D.PAR.03 have been enhanced into generic scenarios. Technical integration concerns the integration of software components that materialise PALETTE services; it elaborates technical issues such as the language used in the service description or standards for service interaction, which have been first discussed in D.IMP.03 and D.IMP.04.

As described in the DoW, the integration process is the key aspect of the PALETTE project. Integration deals with organisational aspects (how users act in their CoPs, in which way they currently use or they are willing to use tools or services to sustain their activity) and technical aspects (how services are designed, developed, enhanced, integrated and interoperate to answer the needs of CoPs members) in an asymmetric way. This is the result of the Participatory Design methodology based on the Actor-Network Theory (see D.PAR.01), where the influences of human and non-human actors are considered symmetrically in order to concurrently design the uses and services in use within the same process.

The conceptual integration part reifies the work of Task 5.4 (Functional specifications of services and scenarios through team work), taking into account the developments of Tasks 5.2 (Software components for integration) and 5.3 (PALETTE services delivery), as well as the guidelines written in the context of Task 5.1 (Standards and Guidelines). The process of designing scenarios is taken a step further with the proposal of generic scenarios. A generic scenario describes a chain of operations and associated functions of PALETTE services, as well as the way these interact to address a generic CoP activity.

The generic scenarios put the emphasis on:

- the identification of interactions between the services in order to sustain the efficient implementation of the associated interoperability;
- the reusability of such scenarios for different CoPs, even though this might necessitate some slight adaptation to suit the specific context and domain of the CoP;
- the usefulness and crucial importance of the activities carried out by CoPs;
- the feasibility of the integration of functions according to the current state of PALETTE services, the capacities of the development teams, and the compliance to standards;
- the capacity to provide evidence of innovation, both at organisational and technical level, brought by the distinctive way they are designed, implemented and validated according to PALETTE principles.

Task 5.1, together with Tasks 5.2 and 5.3, mainly describe the work performed on the technical integration part. More specifically, in Task 5.1 a set of designing guidelines was adopted after the agreement of the development teams. These include common protocols, service descriptions, the overall PALETTE services platform's architecture, and presentation directions. The implementation of the abovementioned platform and the most crucial software components for integration is taking place in the context of Task 5.2. Task 5.3 focuses on the presentation integration, since it addresses the development of components towards the PALETTE services delivery.

The outcomes of this report consist of:

- a methodology to develop, implement and validate generic scenarios, focusing on interactions: interaction between actions of users, between actions and functions, between different functions of different services to sustain the actions, interactions between users and functions through Human Computer Interface;

- an organisation plan to design, implement and validate three main generic scenarios in complement to the already existing six specific scenarios (see D.PAR.03);
- suggestions about which generic scenarios could be the most interesting ones to develop;
- descriptions about the way PALETTE developers support the technical integration at the four different integration dimensions;
- plans for supporting the design and implementation of four integrated fundamental services that enable the provision of search, store, authenticate and notify capabilities to the entire context of PALETTE services;
- suggestions for future work and integration-related issues to be addressed.

1.1 Acronyms

- API : Application Programming Interface
- CAKB : Cross Awareness Knowledge Base
- CGI : Common Gateway Interface
- CoP : Community of Practice
- CRR : Common Resource Repository
- HCI: Human Computer Interaction
- HTTP : HyperText Transfer Protocol
- KM : Knowledge Management
- LDAP : Lightweight Directory Access Protocol
- OAuth: An open protocol to allow secure API authentication in a simple and standard method from desktop and Web applications.
- OpenID : A decentralized single sign-on system
- P2P : Peer-to-Peer
- PDM : Participatory Design methodology
- PI4SOA: A project that provides tools that leverage Pi Calculus to build robust Service Oriented Architectures
- PHP : An open source programming language
- PSP : Palette Service Platform
- PSRF : Palette Service Registry Framework
- REST : Representational State Transfer
- RSS : Really Simple Syndication
- SAML : Security Assertion Markup Language
- SMTP : Simple Mail Transfer Protocol
- SOA : Service Oriented Architecture
- SOAP : Simple Object Access Protocol
- SPARQL : an RDF query language; its name stands for SPARQL Protocol and RDF Query Language
- SSL : Secure Sockets Layer
- SSO : Single-Sign-On
- UI : User Interface
- URI : Universal Resource Identifier
- URL : Uniform Resource Locator
- WADL : Web Application Description Language
- WS-CDL : Web Services Choreography Description Language
- WSDL : Web Service Definition Language
- XML : Extensible Mark-up Language

1.2 Reading conventions

References by author, like (Wenger, 2000), appear in a bibliography at the end of the deliverable. Other references, like (URL: DPP), appear in a “Webography” (list of Web links), also at the end of the deliverable.

2 Conceptual integration - Generic Scenarios to support services interoperability

2.1 Generic Scenarios: the next step in Participatory Design

Generic scenarios are not a new thread of development that would parallel other threads coming from Participatory Design Methodology (PDM). It is the next step to implement PDM after the writing of CoP-specific scenarios according to the report on PDM methodology from WP1.

The first round of scenarios was based on current activities of CoPs and CoPs members, and of their possible support by existing PALETTE services (as they were available during the first year of the project). It was not easy for users, or even CoP delegates or CoP mediators, who were even more aware of the development steps taking place within the project, to figure out how to use these new applications. They were able to think that they could probably use a wiki to create documents, or maybe Amaya for those who were more dedicated to creating teaching supports, or maybe CoPe_it! could be interesting to represent and support debates; but not much more. Then, the PALETTE researchers, within the teams A, B and C, were able to write scenarios which were dependant of the context of each CoP and which emphasize the possible – or sometimes real – use of one (most of the time) or two (seldom) PALETTE Services (as described in D.IMP.03). Moreover, these scenarios, as they were from the early stages of use, were not always a source of innovation for the developers; they were more about incremental improvements – we could speak of "maintenance improvements" – than real development issues.

Thus, we needed to walk a further step in the PD process. We decided to look into building more "generic" scenarios. Such scenarios are generic in several ways:

- they will be less specific on the current activities of the CoPs; we will concentrate on activities which are found within different CoPs (in other words, they are typical for CoPs);
- they will be less influenced by the current state of PALETTE Services; we will focus more on functions available in the PALETTE services, or available in the current tools used by CoPs, or available in other existing tools, which could enable enhancing the support of CoPs activities;
- they will be more detailed, thus they will be able to show the intricacy of different functions of services to achieve composite actions of CoPs members;
- they will take into account the elements that are brought in by D.IMP.03 and D.IMP.04.

Generic scenarios will:

- evidence the characteristic actions, operations, functions[1], services and interfaces required (or necessary or desired) to sustain the life and the development of a CoP;
- evidence the interactions between all actors (human and non-human) that take place for fulfilling these activities and link these interactions with high level purposes at the CoP level;

[1] A function is the logical description of how to achieve an operation (low-level function) or an action (higher-level function). A function is associated to a service, which is a way to practically implement the function. Services can interact (interoperate) through several mechanisms: user action: the user calls a service and then another service explicitly; data transfer (communication): one service is linked to another via a transfer (or sharing) of information(whatever kind of information); direct call: one service can call another service and pass it a set of parameters and/or information; automated process (composite service): an automated process is the chaining of several services according to a registered procedure (explicitly described, possibly customizable) (ex: a workflow).

- suggest possible improvements to better sustain the CoP development by innovating both at the organisational and the technical level (not just help doing what is currently done, but find new ways of doing things or new things that are made possible);
- implement such improvements through innovative architectures and interfaces which take into account the possibilities of technological novelties made available from PALETTE services, the new organisational potentialities opened by the work in CoPs and the learning capabilities of people involved in them.

The reflection about generic scenarios is sustained by:

- the knowledge developed about Communities of Practice; thus, it is important to link scenarios to the high level elements which characterise CoPs, namely: domain, practice, community building, identity building, sense of belonging, etc. (Wenger, 1998; Wenger 2002);
- activity theory as for CoPs activities: activities are driven by an intention, they are composed of actions, themselves composed of operations (Engeström, 1987; Engeström, 1999).

2.2 Generic scenario: how to create value for the CoPs

2.2.1 What is a Generic Scenario in PALETTE?

A generic scenario describes a chain of operations and associated functions of PALETTE services and the way they interact to answer a generic CoP need or activity (intention).

Activities are composed of **actions** that are often organised in chain, being themselves divided in **operations** that correspond to practical, situated and automatic manipulations of tools or machines in order to carry out the actions. So some operations are matched with functions of PALETTE Services. A generic scenario takes into account the motives/intentions which drive the activities and uses of tools in order to sustain the CoPs functioning.

A generic scenario is especially focused on the descriptions of relationship and interactions between users, activities/actions and functions/services, including the key aspects of the user/services interactions through HCI. It is situated, adaptable and provides information about situations where it can be applied.

2.2.2 The value chain representation of generic scenarios in PALETTE

A generic scenario describes typical activities of CoPs, at a granularity level that evidence the role of low level functions, and thus the possible implementation of services and their interactions.

The description of several generic scenarios evidences the existence of some functions which could be mutualised between different services. Such functions may already be present in one or several of our current PALETTE Services (such as e.g. Sweetwiki, Amaya, e-Logbook, CoPe_it!, BayFac, etc.). Each PALETTE Service is a collection of smaller services of different level of granularity, both from a technical and a functional point of view. There must be a choice about which implementation of the service is used as a mutual one. A mutual service (e.g. annotation management, document storage, text editor, graphic mapping representation, etc. which are used by several different kinds of actions) has to be available for calling by any other service independently from its "historical" implementation.

We suggest using a representation derived from the metaphor of the Value Chain (Porter, 1985) as represented in Fig. 1, where:

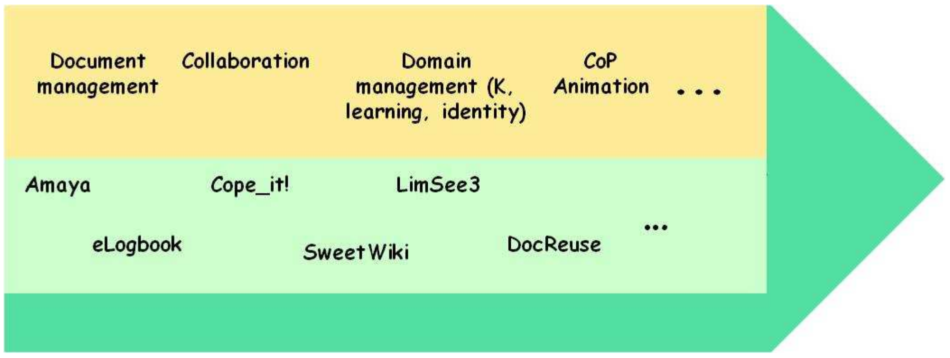


Figure 1 – Generic Scenarios as value creation for CoPs

- the upper level represents the activities of a CoP. We represent here some major typical (generic) activities of a CoP (search for a document, manage CoP members, produce collaboratively a piece of reified knowledge, manage members distinctive competences, debate upon a question in the CoP domain, etc.)
- the medium level represents the functions which are specific to each of PALETTE services in order to support the above activities (representing argumentation, multimedia authoring, reusing structured templates, editing in a wiki, etc.); at this level of granularity, the PALETTE services are represented as entities;
- the lower level represents the mutualised generic functions (or support services) which are needed in order to enable actions which take place very commonly and must be independent from the higher order services; four support services are currently identified: unified access, search engine, tagging/annotating, and notification; a fifth support service enables to bring the full value to the user: this is visual integration, which is responsible to give access to all the other services (specific or support) in the same visual space.

An integrated set of {activity / specific services / support services} is always linked to an intention (or a motive) which drives the CoPs users into action (i.e. the "WHY do I have to do this activity using these services?").

Examples of high level intentions which are specific to CoPs are: identity building, negotiation of meaning, and learning (see Wenger 1998).

It is then necessary to further describe precisely the interactions between all the elements (actions / specific functions / support functions)

2.2.3 Examples of possible generic scenarios

Figures 2-4 show examples of possible generic scenarios represented as above. Naturally, such a representation is not exhaustive and must be further detailed in order to fully understand and describe the related generic scenarios.

<u>Generic Scenario for Collaboration activities</u>

Collaboration actions are the essence of a CoP. It is through collaborative actions that the glue is bind between CoP members, that the motivation is sustained, that the participation takes place, that members are able to shift from peripheral participation towards core activities, that novice members can benefit from experts knowledge, that most of the formal as well as informal learning takes place, etc. Collaboration relies upon functions such as sharing, communicating, debating, decision making, organising collaboration, managing tasks and calendars, etc. Some of these functions are present in

PALETTE services, while some are not. Collaborative functions also rely on the mutualised support services evidenced in other activities, as shown in Fig. 2.

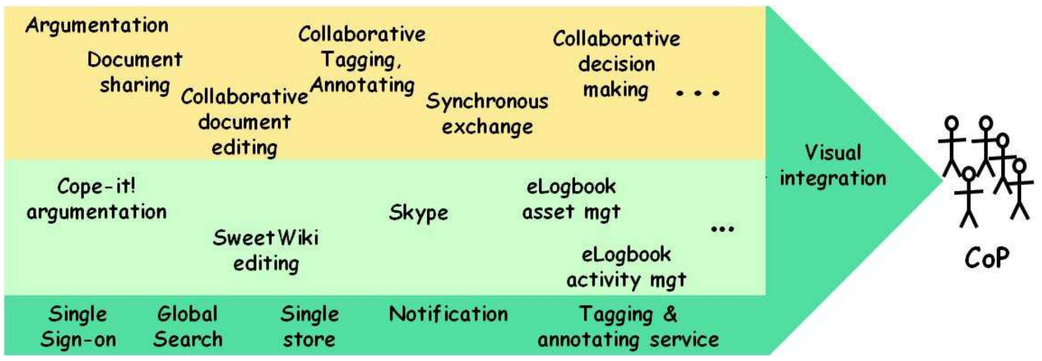


Figure 2 - Generic Scenario for Collaboration actions and functions

Generic Scenario for CoP Animation activities

It is rather commonly agreed that a CoP, beyond its often non formal emerging from more formal context, and despite the needs to remain a rather informal network of people and activities, strongly benefits from an animation role to sustain the development of its life-cycle.

CoP animation gathers actions as diverse as managing members, managing members profiles and competences, capitalising on CoP history and building the CoP memory, managing events (internal and external), providing space and time for knowledge capitalisation, facilitating the CoP life, sustaining members' motivation, valuing everybody's contribution, evidencing the value created within the CoP and provided by the CoP to its environment, etc.

Besides the ones which were already described in the preceding scenario, CoP animation mobilises specific functions like managing files for members and profiles, advising and managing events, communicating inside and outside the CoP (publishing and disseminating functions), and knowledge based functions (history and memory support). It also uses the mutualised support services, as shown in Fig. 3.

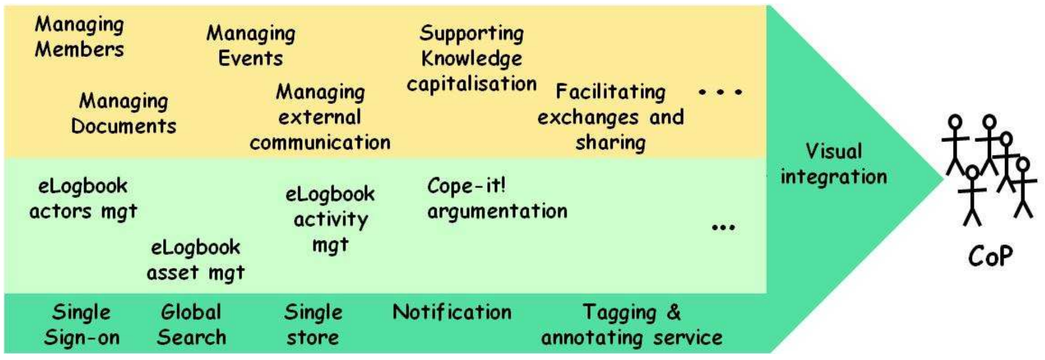


Figure 3 - Generic Scenario for CoP Animation actions and functions

Generic Scenario for Domain Management and Knowledge Reification activities

Domain Management regroups all the activities related to the domain of the CoP, i.e. the distinctive practice that gathers people together. In PALETTE, we have CoPs who belong to the Education area, but each of them has its own domain; for example, Did@ctic gather teachers and future teachers who are interested in improving practice in successfully implementing innovative pedagogical systems including information and communication technology, whereas @pretic gathers school teachers whose activity is evolving as resource persons to support the introduction of information and communication technology in school; even though these areas are related, the two CoPs have distinct domain of practice.

Domain Management comprises all actions related to the management of knowledge and competencies related to the domain, the learning and training activities which enable the improvement of both individual and collective knowledge and practice, the knowledge reification, sharing and publishing activities, etc.

Domain management also includes all the activities related to knowledge reification, i.e. those related to document management. Here, “document” refers to an object which might range from a single sheet of text -paper or electronic- to composite, multimedia, complex in structure documents with their history, their archived versions, the different authors, the annotations, and all related metadata. Creating, storing, and retrieving are the basic actions performed on such documents, together with editing, annotating (and tagging) them. More complex operations can be envisaged when it comes to reusing, restructuring, and recomposing some documents to produce new ones, for example. Furthermore, all these actions can be done by individuals, but also collaboratively.

Domain Management strongly relies on KM and KM related functions, such as managing ontologies, managing semantic resources and applications, managing representations of knowledge, sharing and publishing of reified knowledge, etc.

Again the mutualised support services are used in this area, as shown in Fig. 4.

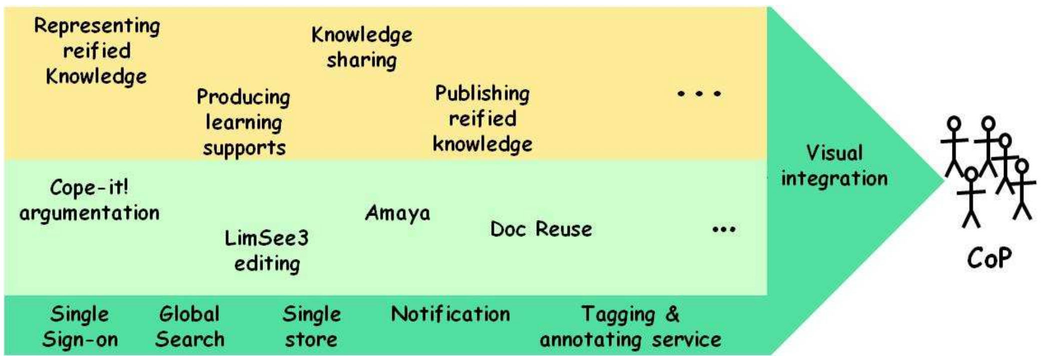


Figure 4 - Generic Scenario for Domain Management actions and functions

2.2.4 Which generic scenarios for PALETTE?

This list of generic scenario is not exhaustive, though we think it pretty much covers the fundamental issues around CoPs life and activities. Naturally, there are some overlaps between the examples of generic scenario described above, as different activities are often relying on common actions and functions.

Thus, we decided to focus the next steps of Participatory Design on the design and implementation of:

- a generic scenario for domain management and knowledge reification focused on resources reification;

- a generic scenario for collaboration focused on debate and collaborative decision making;
- a generic scenario for CoP animation focused on CoP identity building, and
- the support services which are evidenced in the different generic scenarios.

As regarding support services, we decided to focus primarily on four support services: Single Sign-on, Global Search, Single Store and Notification of Events.

2.3 The process of designing generic scenarios

The outcomes of the process of designing generic scenario are:

- a description of a set of activities performed by CoPs and CoPs members according to their intentions;
- a description of the functions related to each granular action (or operation depending on the level of detail which is necessary for the scenario designers to fully understand, design and implement the scenario);
- a description of the different levels of interactions, and the mechanisms of these interactions (according, for example, to the findings of D.IMP.03 and D.IMP.04), including the HCI aspects;
- a representation of the scenario.

Designing a generic scenario implies considering two “deconstruction” threads followed by a “reconstruction” one, and the development of the necessary services:

- deconstruction of CoP functioning following an activity-based analysis:
    - selection of CoPs **activities** that are typical (and preferably common to several CoPs); it is to describe each activity taking into account the CoP as a specific organisational context;
    - decomposition of these activities into **actions** that CoPs members perform;
    - if needed, decompose these actions into **operations**; the degree of granularity of the decomposition in actions and operations is strongly linked to the specificity of needs and behaviour in a CoP, and also to the granularity level of software functions that will have to implement them.

- deconstruction of PALETTE services into granular functions:
    - identification of existing **functions** in PALETTE services or in other available tools, which could be used to perform each action or operation;
    - identification of functions that do not correspond to defined actions or operations, but that can be useful to define new ways supporting an activity;
    - matching between actions-operations and functions.

- reconstruction of the generic scenario, comprising:
    - identification of common functions which should be mutualised into support services;
    - construction of chains of actions-functions which evidences the necessary interactions between the different functions;
    - identification of and description of direct communication between services (service-service interaction) which are necessary or can be useful to perform the actions;
    - identification of user-level actions which correspond to direct access to services by user, give hints about ergonomic recommendations for the user interface and cognitive constraints from other applications commonly used within CoPs;
    - description and representation of the generic scenario.
    - validation by the teams

- design and implementation of the services which will implement the functions (specific and support).
- trialling and validation.

Examples of the deconstructions steps can be find on the SwikiPALETTE space at the following addresses:
http://argentera.inria.fr/swikiPALETTE/data/TaskForce/GenericScenarioExample1.jsp
http://argentera.inria.fr/swikiPALETTE/data/TaskForce/GenericScenarioExample2.jsp .

2.4 Organisation of the process of designing generic scenarios

2.4.1 A task force to re-align the Participatory Design Process

In order to prepare this work, we decided to constitute a taskforce within WP5. This task force was in charge of:

- identify generic scenarios to be developed;
- reorganize teams around these scenarios;
- propose a methodology and a template for the development of generic scenarios (and the related integrated services);
- propose a work plan for teams taking into account the elements of the evaluation of the PDM.

This task force was composed of pedagogical and technological partners. It worked from October to December 2007, both in virtual and face-to-face meetings. The work and findings were collaboratively reified using the SwikiPALETTE implementation of the SweetWiki PALETTE service. and presented to the project members at the plenary meeting in Lyon (December 19-21, 2007).
(http://argentera.inria.fr/swikiPALETTE/data/TaskForce/TaskForceHome.jsp)

2.4.2 An operational organisation to realise the generic scenarios

The generic scenarios are designed, implemented and validated using a similar organisation with the one used to develop the specific scenarios, i.e. in collaborative teams regrouping CoPs mediators, service mediators and other PALETTE researchers.

The teams are in charge of following the process described above (see 2.3 - The process of designing generic scenarios):

- describing activities / actions / operations taken from the specific scenarios;
- describing functions taken from the PALETTE services description;
- describing the matching between actions/operations and functions;
- identifying support functions that should be mutualised;
- describing the chains of actions/operations – functions;
- identifying and describing the interactions between functions (according to D.IMP.04);
- identifying the ergonomic and cognitive constraints which will shape the user interface;
- write the generic scenario;
- develop the functions / services necessary to implement the generic scenario;
- implement the generic scenario, and
- validate the generic scenario.

2.4.3 Choice of scenarios to be developed

On order to facilitate the work of teams, and to accelerate the process, it was decided to focus on three main scenarios, which are subsets of the generic scenarios described above:

1) Reification scenario.

This scenario is a subset of the Domain Management and Knowledge Reification. It can be derived from the specific scenarios of Did@ctic and Apretic, for example. It is based upon the following:

Activities/actions	Functions / services
Production of resources (e.g. documents) Enrichment of resources with semantic information Search for existing resources Reuse and transformation of existing resources Building of the CoP memory	Sweetwiki DocReuse Amaya, Limsee3 Semantic FAQ, BayFac Support services: unified access, global search, tagging and annotating, notification, visual integration

2) Debate and Decide scenario.

This scenario is a subset of the Collaboration scenario. It can be derived from specific scenarios like LearnNett of Adira.

Activities/actions	Functions / services
Debate about an issue Argumentation using internal and/or external resources Collaborative decision making Awareness about the above activities within the CoP	CoPe_it! (argumentation) Link-widget e-Logbook (asset management) Support services: unified access, global search, tagging and annotating, notification, visual integration

3) CoP animation.

This scenario can be derived from scenarios like Adira, for example.

Activities/actions	Functions / services
Management of CoP collaborative activities Management of members Management of events Management of resources related to members / events / activities Awareness about the above activities within the CoP	e-Logbook actors management e-Logbook asset management e-Logbook activities management editing functions (SweetWiki, Amaya, etc.) CoPe_it! Support services: unified access, global search, tagging and annotating, notification, visual integration

2.4.4 Schedule of work

It is important to understand that the development of services is not a consequence of the scenario design; it is **included in the scenario design, development, implementation and validation.**

The foreseen schedule is the following:

task	**time**
Designing an outline of the generic scenario with emphasis on: • which are the functions already existing in services • which are the support services (common services) necessary • which improvements are needed to ensure the integration of services	February - March 08
Refinement of the specification and first stages of implementation of the three generic scenario, including: the identification of necessary incremental developments of existing services the modular (individual) tests of functions the focus on integration, interoperability and interfaces design, development and tests	March – June 08
Further stages of implementation and validation of the scenarios according to the validation principles of PALETTE together by the teams and the CoPs	June – November 08

2.4.5 Proposal for the reorganisation of teams

Three teams will be formed to implement the above scenarios (or part of them). The organization of these teams is described below.

Team 1 – Knowledge Reification	
Developers	SweetWiki, DocReuse, Amaya, Limsee3, Semantic FAQ, BayFac
CoPs	Did@ctic, LearNett, ePrep, Adira, Apretic, Form@HETICE, TFT, TICFA, TICEF,CoPeL, APCDE
Examples of Implementation	
1. Documents produced with Amaya, or DocReuse are imported and annotated in SweetWiki.. CoPs members search for documents using a global search engine. CoPs members could reuse existing documents to produce new ones. 2. Emails are annotated and CoP members could search for relevant emails. 3. Documents produced with Amaya, SweetWiki etc. are indexed and classified by BayFac. Documents are then searched to be reused.	
Team - Debate and Decide2	
Developers	CoPe_it!, Link-widget
CoPs	Didactic, LearNett, Adira, TGC, TICFA, TICEF, CoPeL
Examples of Implementation	
1. CoP members annotate their discussions in CoPe_it! using Link-widget and could search for relevant discussions. 2. To enhance their argumentation, CoP members could search for resources using the single-search and import relevant documents, mails….	
Team 3 – CoP Animation	
Developers	E-Logbook, CoPe_it!, SweetWiki
CoPs	ePrep, Adira, Aradel, TICFA, TICEF
Examples of Implementation	
1. CoPs members import documents from SweetWiki or discussions from CoPe_it! in e-Logbook 2. Each member edits their own page and profile (content with competencies, etc., not administration profile) in e-Logbook 3. Members create collaborative documents with SweetWiki within a collaborative activity of e-Logbook 4. Each member creates activities in e-Logbook and invites other members; an activity may be to join a discussion in CoPe_it! 5. Members can manage a schedule of events and activities. Accounts of activities/events, created with SweetWiki are managed as assets in e-Logbook	

Each team is composed of developers (T), CoPs Mediators (P), and possible other PALETTE researchers (T,P). Developers propose a set of “scenarios instantiations” which will be validated by the Mediators. Then the design in use process goes as follows: developers propose prototypes that will be tested by CoPs. The coordinators ensure that the development is done according to WP5 requirements and guidelines and report on PDM methodology from WP1.

3 Technical integration

3.1 General issues: Definition and Integration types

In the context of the PALETTE project, the term “technical integration” can be defined as the way the PALETTE partners support the provision of integrated functionality to the CoPs. In particular, technical integration in PALETTE concerns the adoption of relevant standards and development guidelines that facilitate interoperability of services, as well as the implementation of appropriate software components for integration and delivery of the PALETTE services. The guidelines apply to different dimensions of integration, each of them leading to specific design choices that are described below.

3.1.1 Dimensions of integration

The classification of PALETTE services presented in D.IMP.04 introduced a distinction between what we called Web Services and Interactive Services. Web Services have no user interface and interact with other services, while Interactive Services have a graphical user interface and interact with the user.

An integrated functionality can be available to CoPs through different ways: some integrated compound Web services that can provide information concerning the entire context of PALETTE, or some integrated interactive service that present integrated user interfaces of the PALETTE services.

The present situation in the PALETTE project is that several interactive services are available as full blown Web applications, such as CoPe_it!, e-LogBook, SweetWiki and DocReuse, or as full blown applications such as Amaya or LimSee3. These interactive services are currently monolithic. They are not yet "reusable" as bricks to compose new interactive services.

This does not prevent some simple forms of integration, in a way that users can interact with several of these services at the same time. For convenience, they can launch all these services in different tab views of the same browser window, or in different windows. However, such “manual” integration does not support advanced interactions between services (such as those defined in D.IMP.03). These interactions have been divided into three levels: transmission of data and metadata between services, direct call of a function from one service into another one, and finally composition of several service functions. Without modifying the current interactive services, the only interaction between services which is supported by the browser environment is limited to the cut and paste of data from one service to another one. This is far from sufficient to support the kind of interactions between services that are envisioned in the scenarios and will be defined when turning the generic scenarios into concrete scenarios.

To proceed further with integration of services and to allow interaction between services on the three levels defined in D.IMP.03, PALETTE developers are required to redesign the tools and their interoperation methods not only from a user interface point of view but also from an internal view in which information storage and representation, as well as service availability, are taken into consideration.

To summarize, the integration between the existing PALETTE services needs to be analysed at the following dimensions:

- *Information integration* in which information created from one PALETTE service can be transformed and imported to another PALETTE service;
- *Communication integration* that can assure that every network communication between tools and services can be established with a specific way;

- *Platform Integration*, in which a common Platform is the basis for the publication of the existing services, registration of the new ones, and discovering of the PALETTE offered service;
- *Presentation Integration*, in which users can access several PALETTE services through a unique interface or can use different PALETTE services in a common way.

These dimensions of integration will be elaborated in a dedicated section below.

3.1.2 Approach of technical integration in PALETTE

The diversity of the available PALETTE services, as well as the creation of new services in the context of PALETTE, make functional integration not an easy task:

- Existing PALETTE services are operating under different platforms; desktop applications, Web applications, Web services etc. co-exist in the PALETTE framework.
- CoP requirements are growing up dynamically during the usage of the PALETTE services; the difficulty to predefine useful scenarios of using integrated functionality leads developers to design open techniques in order to easily support new requirements.

To further proceed with integration of services and to allow interaction between services on the three levels defined in D.IMP.03, we have started different types of actions. First, some experiments have been done to develop ad-hoc pair-wise integration of interactive services. This method allows to design interactions between services that match exactly one scenario, for instance to directly open a SweetWiki page into Amaya and save it back into SweetWiki. However, this is a costly approach that requires a direct intervention on the applications. It is possible to do it as long as the applications are in a design in use phase.

Second, we have started to define some Web services to expose the functionalities of all interactive services, following the guidelines defined in D.IMP.04. This approach is the most ambitious as it will allow the development of new interactive services in the future based on the existing Web services, without having to modify the original applications. For instance, this is a necessary step to be able to develop miniaturized versions of the current applications, implementing subsets of their functionalities, to be integrated as widgets into the composable PALETTE portal, or eventually, as widgets into other interactive services. This is also a necessary step to be able to compose several functionalities in the existing applications and propose new composite services or new mashups of services. We have already defined the PALETTE Service Platform architecture with the purpose of supporting the discovery and the execution of such services, as described in D.IMP.04.

During the definition of the Web services derived from each interactive service, we have been able to identify four groups of services, that we call *support (or fundamental) services*. By matching these support services with the needs expressed in the scenarios, we have discovered that each of the support services could be potentially composed in different ways within all applications to offer a greater integration within PALETTE applications. The support services and the requirements for their integration derived from the scenarios are shown in the following table.

Support (or fundamental) service	**Requirements (U = User)**
Identification	*U wants to share her profile information between services because she does not want to make multiple user accounts within PALETTE applications, or at least she wants to input her profile information only once. U wants to avoid signing in several times when navigating from one service to another in a single session.*

Search	*U wants to find a resource that has been embedded or referenced into a service, or a resource that has been created in a service such as a page, and thus without knowing in advance in which service(s) it is. U wants to use different criteria to find a resource (by name, authors, date, keywords, etc.).*
Storage	*U wants to store a resource in one service and to be able to embed or to reference it from any other service without having to upload it again into that service.*
Notification	*U wants to be notified of events (creation of resources, online presence of CoP members, etc.) that happens in any service with having to login into each service.*

The first three support (or fundamental) services counterbalance the fact that information and knowledge created in PALETTE services is locally interpreted and stored.

From this point, we now envision the integration in PALETTE as a continuation of pair-wise integration efforts for some specific integration scenarios. But more importantly, we now concentrate on a careful design and selection of APIs for the support services that will ease the creation of compound services in a service oriented context. In doing so, we are in line with similar approaches that have been proposed by the main vendors of social network platforms to augment interoperability between them and which are underway, such as the OpenSocial API (URL: OSG), or the public group DataPortability.org (URL: DPG).

Our first results with the identification, search and storage integration are described in a dedicated section below. The notification support service will be elaborated in a future deliverable.

3.2 Supporting Integration in PALETTE

This section presents the achievement of integration in PALETTE. In particular, it describes how PALETTE developers adopted common standards, created a common platform and customized their PALETTE services towards this target. The following integration analysis is structured according to the aforementioned classification of the technical integration types (information, communication, platform and presentation).

3.2.1 Information Integration

The initial step to support information integration was the focusing on data integration using XML formation. Integration of heterogeneous data sources is a complex activity that involves reconciliation at data models, data schema and data instances (Bertino and Ferrari, 2001). XML in PALETTE comes to assist the addressing of these issues by describing common information models, declaring global schemas and finally by dressing information content while it is being interchanged or interpreted.

Apart from the above common definitions of models and schemas, some software components for tagging or adding additional information over valuable resources were already developed within PALETTE. Generis, CAKB, and SweetWiki are some of them.

Concerning finding and accessing of documents, the PALETTE framework will support a common document storage repository in which different PALETTE services will be able to store, find and retrieve documents. By this way, it is possible not only to create a global storage place for CoPs' resources, but to allow interchanging of resources within PALETTE. Depending on the resource data model and schema, it is possible for each PALETTE service to provide specific import-export converters in order to "translate" a resource coming from another PALETTE service to a readable resource and vice-versa. Examples of such converters have been elaborated between CoPe_it! and e-Logbook (e.g. transformation of a CoPe_it! workspace to an e-Logbook project). Tools for semantic searching of available documents are also provided within PALETTE (e.g. BayFac and Corese semantic search engine).

3.2.2 Communication Integration

The adoption of common protocols and guidelines regarding communication between PALETTE services plays a crucial role in the technical integration. In PALETTE, developers made an agreement on the usage of the Web Service methodology as the common way to provide functionality to other tools and services. Thus, the Simple Object Access Protocol (SOAP) together with the Representational State Transfer (REST) are the common accepted protocols for intercommunication within PALETTE.

For each provided PALETTE service, developers are responsible to generate and publish a specific service description that analyzes in detail all possible ways to establish connection with them, and invoke the available methods by passing the appropriate parameters. Parameters definitions are also presented to the descriptions. These descriptions are mainly in the format of Web Application Description Language (WADL) and / or Web Service Description Language (WSDL).

Most of the messages being sent are in the form of XML or simple attribute-value pairs. Apart from the above communication guidelines, the addition of the RSS protocol to provide data sources and notification event lists has been agreed, as well as the usage of SMTP as a means to provide low cost and ubiquitous (aka email) user interface UI to some services.

3.2.3 Platform Integration

PALETTE developers are following the direction towards platform integration by designing and implementing a Service Oriented Platform called the PALETTE Service Platform (PSP). However, apart from the service orientation, there were also some ad-hoc pair-wise implementations of interoperations between PALETTE services that are also presented below.

The PALETTE Service Platform (PSP)

The PALETTE Service Platform focuses on the use of the PALETTE as an open infrastructure where services and tasks can be defined, composed, and enacted in a fully customizable way. In order to reach this level of platform integration, PALETTE developers made an agreement about the notion of the *PALETTE Service*, the architecture that will guide to the platform creation, and the methodology that must be followed whenever a PALETTE service registration, publication, discovery or invocation action takes place.

Below, a brief description of PSP is given (a detailed description can be found in D.IMP.04).

The PALETTE Service

The PALETTE Service Platform tries to provide an integrated mechanism for creating, handling and using PALETTE Services. We consider PALETTE Service as the fundamental entity of the architecture. A PALETTE Service can be defined (in a technical perspective) as the entire set of PALETTE software components that can be useful for CoPs. These include: web services, PALETTE tools, Web applications and composite services. Apart from the above, some external or 3rd party services can also participate to the platform.

The Platform Architecture

The PSP architecture is based on both a data mashup architecture and a service oriented one. Its fundamental modules are communicating with each other via REST or SOAP protocol. These modules are (Figure 5):

- *The Service Delivery:* it provides a User Interface for Service Registration and Discovery and a Visual Integrated Portal for accessing PALETTE Services through widgets. Also, it includes a Cross Awareness Knowledge Base (CAKB) for supporting awareness of activities within CoPs.
- *The PALETTE Service Registry Framework (PSRF):* a Service Registry and Access and Identity mechanism that allows the registration, publication and discovery of service descriptions. Service descriptions are in the form of XML according to an integrated service description XML schema.
- *The Service Orchestration:* an engine that allows the creation of composite services from existing ones.

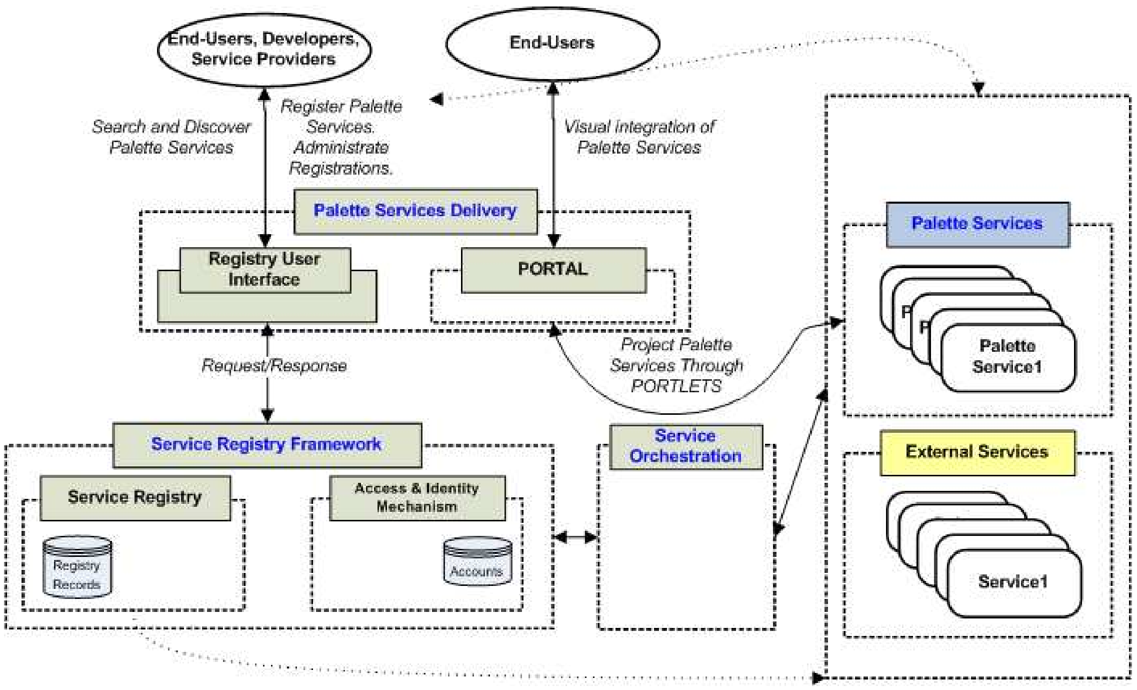

End-Users, Developers, Service Providers
End-Users
Search and Discover Palette Services
Register Palette Services. Administrate Registrations.
Visual integration of Palette Services
Palette Services Delivery
Registry User Interface
PORTAL
Request/Response
Project Palette Services Through PORTLETS
Service Registry Framework
Service Registry
Registry Records
Access & Identity Mechanism
Accounts
Service Orchestration
Palette Services
Palette Service1
External Services
Service1

Figure 5 - The overall architecture from a design perspective.

Using the PSP

Potential users of the proposed integrated platform are humans or software that can be facilitated by the usage of the PALETTE services. They can be classified to: end users, developers, PALETTE Service Providers, PALETTE services and external Applications or Services.

Current status of the PSP development

The common XML schema of the PALETTE service description document is available at [http://150.140.18.39:91/psrf/img/ServiceDescriptionSchema.xsd]. The PSRF that includes Service Registry and Access and Identity has been developed and became operational. This module is available at [http://paletteregistry.cti.gr].

The PALETTE Service Portal implements the w3c widget specification draft with minor extensions. This software is currently in beta version. Future work steps include the development of an inter-widget communication subsystem enabling the creation of composite Web applications as a mashup of PALETTE widgets, and the integration of a single sign-on feature. Regarding the CAKB, an RSS 1.0 extension has been specified. It permits to follow the operations made on documents in various Palette services and acquire meta-data on these documents. All this meta-data is centralized in a Generis knowledge base, which can be exploited by two means: by a REST Web service interface and by a BayFac faceted search interface. For the end-user, it can already be used as a meta-data based cross-service search engine. Future work steps on the CAKB will cover the development of extended RSS feeds on various Palette services, integration with the keywords-based cross-service search engine, and integration with a single-store repository.

The work on service composition/orchestration is in progress. After the initial study of current implementations of engines, a Pi4SOA composition framework has been installed. Based on Pi4SOA, the composition case of the "global-search-service" was tested.

Future work

Future work regarding PSP can be classified into four different tasks:

- Finalization of the development of all remaining PSP subsystems;
- Customization of the subsystems in order to achieve an optimal level of communication between them;
- Creation of the appropriate widgets for the Portal (for each PALETTE service that is required to be visually integrated);
- Filling of the PALETTE Registry through its UI with the PALETTE service description documents for each of the available PALETTE services.

After the completion of the above tasks the PSP will be ready to be used as an integrated platform for both CoPs and PALETTE developers.

Ad-hoc, pair-wise combination of PALETTE services

Although the PALETTE project has adopted a centralized approach to service integration via the Palette service platform (PSP), for a number of interoperability examples an ad hoc integration approach has been adopted. In the context of tool interoperability, the notion of ad hoc is used to refer to the situation where two tools explicitly interoperate without being aware of the Palette service platform and resolve all integration related issues (including the semantics of operation and data) in a bilateral way. The basic reasons to adopt such an ad hoc approach are:

- *Ease of implementation:* Although powerful, PSP introduces overhead in the development process since all PALETTE services must comply with the adopted interfaces and guidelines. Moreover, PSP is still under development which currently hinders developers to experiment with PALETTE service interoperability. Adopting an ad hoc interoperability approach lessens the development efforts and aligns well with methodologies of rapid prototyping of PALETTE services.

- *Tight integration between* PALETTE services*:* Since in an ad hoc approach no more than one additional party is involved, semantic issues can be resolved in greater detail leading to more usable services.

- *No commitment:* The ad hoc integration does not require commitment to a plethora of guidelines, thus making integration efforts more flexible.

Adopting an ad hoc integration approach has also shortcomings, the most important one being that any related integration decision is difficult to be generalized. The lack of a common framework harms significantly the maintainability of the PALETTE services and makes the provided solutions prone to errors.

Scenarios of the achieved experiments

Some interoperability examples that follow an ad hoc approach are presented below.

- **e-Logbook ↔ CoPe_it!**

This scenario concerns how e-Logbook's Context-Aware View is called from CoPe_it! and what can be seen from this view as the user switches from one focal element to another. A detailed description of this p2p implementation can be found at D.MED.04

- **e-Logbook ↔ CoPe_it!: Transparent profile synchronization**

Another scenario of interaction between CoPe_it! and e-Logbook aims at satisfying the CoP requirements of sharing identify and profile information. In this scenario, users will not have to fill the same profile information more than once. Further implementation details of this example can be found in Appendix A.

- **SweetWiki ↔ CoPe_it!**

The overall goal of this integration effort is to augment existing services of CoPe_it! by utilizing the ontologies (and basic KM services) that have been developed in the context of PALETTE. In particular, future releases of CoPe_it! will permit the annotation of individual resources found in its collaboration workspaces by CoP members (as well as annotation of entire workspaces), in order to explicate the role these resources play within the problem domain. Further implementation details of this example can be also found in Appendix A.

- **Amaya ↔ SweetWiki**

Interoperability methods between Amaya (v10) and SweetWiki can be used in order to support on-line editing of information resources in different contexts by both tools. Experiments towards this particular scenario were took place by both involved partners. For this scenario, a unique referencing mechanism through URIs, together with some import routines were developed in order to support downloading from and uploading to SweetWiki. Further details can be found at D.INF.05.

Future work

Concerning the aforementioned examples, developers are focusing on the completion of the ad-hoc modifications in order to provide stable bindings between the associated tools. However, all the scenarios above are being re-examined towards service orientation in the context of PSP. Future work may be required in case that another specific interoperation scenario requires ad-hoc implementation.

3.2.4 Presentation Integration

The presentation integration aims at increasing the perceived continuity between services when they are manipulated at close time interval, or when they are combined to perform integrated actions. Examples of combination include the user searching for an activity in e-Logbook tagged with the same keywords which are used to tag the SweetWiki page she is currently viewing.

The definition of guidelines for homogeneous user interfaces, as proposed in D.IMP.04, is a way to create a consistent user experience when navigating from one service to another. This is a basic form of presentation integration that must be done in conjunction with usability recommendations as the visual style, visual layout and graphical behaviour contribute to the usability of the applications.

The second form of presentation integration that we have started to investigate is the visual integration of several user interfaces into the same screen space. This can be achieved by using different methods that will be described in a future deliverable (D.IMP.06). In summary, there are two main approaches which have some overlaps. The first approach is a centralized approach. It is based on a *Portal*, which is a dedicated interactive service specifically designed as a container for visually integrating some miniaturized versions of subsets or of full interactive services, into the same window. The second approach is a distributed approach. It is based on the provision of specific pieces of user interfaces that can be embedded into existing services, to combine them with one or more external services. Both approaches allow the visually integrated services to interact together. For instance, one service can provide data to the other service. This requires some integration beyond the presentation layer, to share data or to export functionalities directly from within the browser rendering the user interfaces.

Future Work

The presentation integration solutions suggest some evolutions of the Portal container API and the widget API that we have started to investigate (these will be described in D.IMP.06). Currently, the following directions are considered as the most promising ones to open up innovative visual integration instruments:

- following an approach similar to the JavaScript Client Library for Facebook (URL: JCL), which allows to create a runtime environment to execute Facebook widgets in any application, we could imagine a solution to allow Portal widgets to be embedded into any PALETTE Web application. This would augment the motivation to create Portal widgets as they could be then reused in pairwise integration scenarios.
- the visual integration by itself is not sufficient to allow complex interaction between widgets or between a widget and its container application; thus, we plan to study some possible client-side inter-widget communication protocols that would allow to facilitate the construction of client-side mashups.

Another ongoing work that will be described in D.IMP.06 concerns the functional integration of the notification services. We are currently studying the possibilities to provide notification transport layers through the Cross Awareness Knowledge Base.

3.3 Providing support/fundamental services for integration according to generic scenarios

This section describes proposed solutions about the supporting of the four support/fundamental services for integration (i.e. global “cross-tool” search, single sign-on, single store, and notification of events) within the PALETTE project. The implementation for each given solution has already started and will be completed in the context of WP5.

3.3.1 Global “cross-tool” Search

<u>Composite service «Global Search Tool» principles</u>

The Global Search Tool is a composite service, which aggregates individual search Web services (SOAP or REST) provided by some PALETTE services. It acts as a query dispatcher to these services, allowing a single access to data and information handled by different PALETTE services.

This case of composite service has been used to test the different manners of making a composition, and in particular to make a decision concerning the relevancy of using composition frameworks and a composition engine in the PALETTE context. Two kinds of approaches have been tested:

- a direct implementation coded in PHP, and
- a choreography implementation based on PI4SOA (URL: PIS)

Identified search services

The following list presents identified PALETTE services providing a search function:

- CoPe_it! (more information in http://copeit.cti.gr/site/doc.html) (URL: MED09)
- e-Logbook (more information in http://copeit.cti.gr/site/doc.html) (URL: MED09)
- Corese/SeWeSe
- BayFac
- PALETTE Learning Platform (AnaXagora-LMS)

As summarized in the table below, only three PALETTE services have been actually used for the implementation test, as the others were not yet accessible or available during the test.

PALETTE Service	**Existing WS**	**Web service implemented in Search Engine**
e-Logbook (REST ws)	**Query for activities list:** returns activities for the authenticated actor (name, description, ID and member actor and roles): http://e-Logbook.epfl.ch/spaces.xml **Query for assets list:** returns deliverables for the authenticated actor (name, description, ID, submission deadline, validation deadline, and the parent activity): http://e-Logbook.epfl.ch/assets.xml **Query for deliverables list:** returns deliverables for the authenticated actor (name, description, ID, submission deadline, validation deadline, and the parent activity): http://e-Logbook.epfl.ch/deliverables.xml	Creation of a Web service in the search engine (entry: keywords, output: list of activities, assets and deliverables in RSS format As for the moment the search Web service isn't implemented on e-Logbook, this Web service (on the search-engine) firstly queries the list of activities, assets and deliverables for the authenticated user (authentication is realized directly in the Web service, with test credentials), and secondly tests if the found elements correspond to keywords (according to their label and description).
BayFac (REST ws)	**Search Web service:** **Input:** list of keywords, "," separated, **Output:** list of URI of found instances **Web service address:** http://citi-efficient2/generic/index.php /rest/fs/Document/instance/?keywords=a,b	Creation of a Web service in the search-engine taking as entry the list of keywords (";" separated), and returning in RSS feeds the list of found instances. As the keywords search Web service returns a list of URI of found instances on BayFac, this Web service (on the search-engine) also searches for detailed information of the found instances in order to output a complete RSS feed (with title, description and link of found instances).
PALETTE Learning Platform (SOAP ws)	**Search Web service:** **Input:** • **Title:** search realized on the module title • **Author:** search on the author's name • **Keywords:** keyword search realized on the module title, chapters titles, module's keywords and description **Output:** • **Return:** an xml formatted string containing for each module found its title, description and link **WSDL:** http://www.anaxagora.tudor.lu/PALETTE/LCMS/Web service/ws_PALETTE.php?wsdl	Nothing had to be added in the search engine, except the Web service call.
CoPe_it!	**Search Web service:** **Input:** a matching string (query) and a set of parameter options. **Output:** for each matching resource, its unique identification, along with its title is returned. **Exception:** Exception is thrown if parameter options are not in a valid form.	The Web service didn't work at the moment of the implementation and so isn't used yet in the search engine.
Corese / SeWeSe	**SPARQL Web service:** **Input:** sparqlQuery is a string containing a SPARQL query **Output:** a string in SPARQL format **Exception:** • EngineNotFoundException thrown when engineID does not denote an engine • MarlformedQuery thrown when the query does not correspond to the SPARQL format • QueryRequestRefused thrown if the engine cannot respond to the query	The Web service didn't work at the moment of the implementation and so isn't used yet in the search engine.

Service structure

Figure 6 illustrates the structure of the service that has been implemented, which has the following interface:

- **Input:** list of keywords, “;” separated,
- **Output:** an RSS channel, which has the following format

```
<rss version="2.0">
      <channel>
            <title/>
            <description/>
            <link/>
            <item>
                  <title/>
                  <description/>
                  <link/>
           </item>
    </channel>
</rss>
```

The choices of input and output format have been made in an ad-hoc manner, as the purpose was to test a composition implementation, and there were no specific constraints. The three Web services used are those of e-Logbook, BayFac and the PALETTE Learning Platform. They are called in parallel.

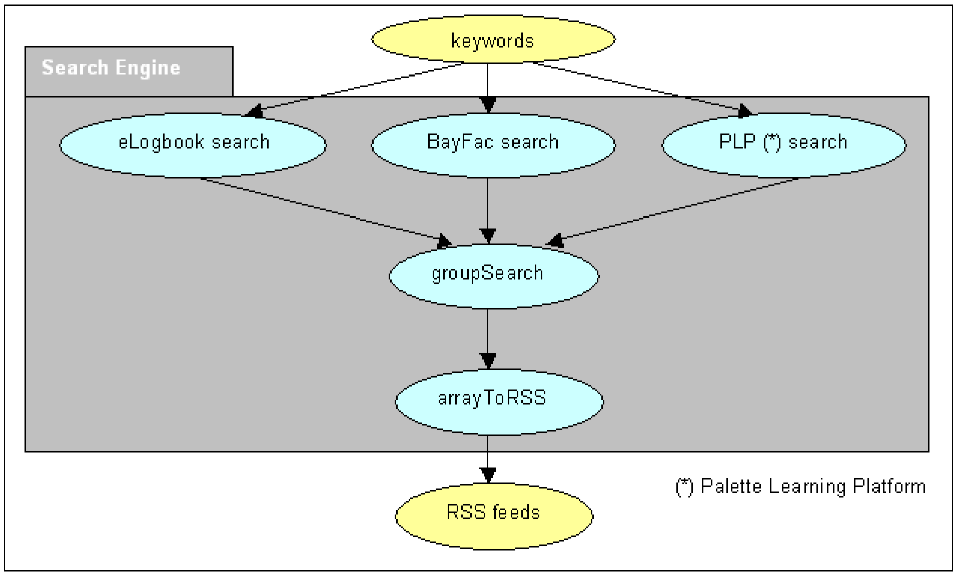


Figure 6 - Search Engine scenario

Direct implementation

According to the current state of the PALETTE services used, the implementation of the composite service implies calls to different functions to get the wanted description of items retrieved from the keyword request. Whatever the method used to create the composite service, be it manual or using a composition framework, the different steps detailed in Figure 7 are necessary.

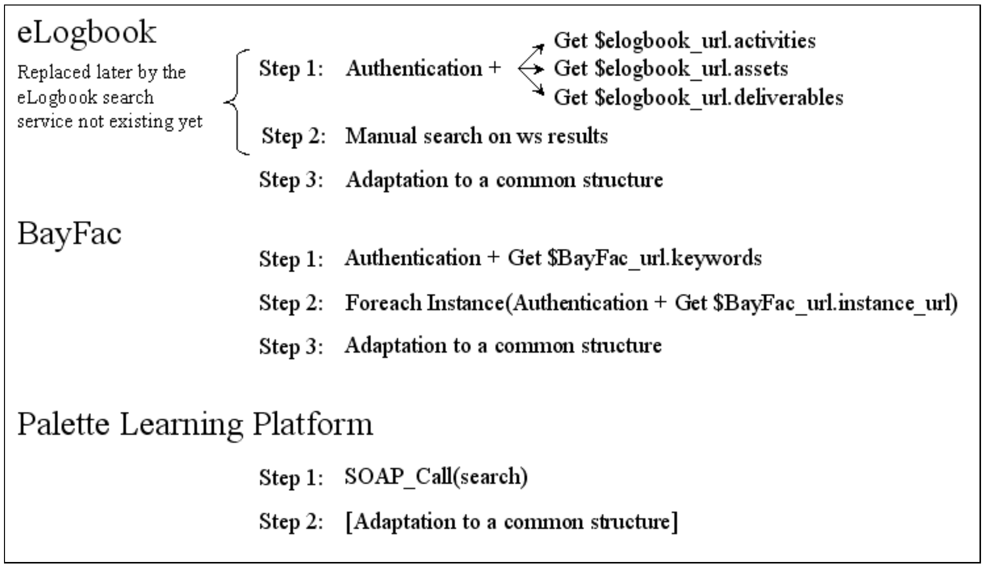

eLogbook
Replaced later by the eLogbook search service not existing yet
Step 1: Authentication +
Get $elogbook_url.activities
Get $elogbook_url.assets
Get $elogbook_url.deliverables
Step 2: Manual search on ws results
Step 3: Adaptation to a common structure
BayFac
Step 1: Authentication + Get $BayFac_url.keywords
Step 2: Foreach Instance(Authentication + Get $BayFac_url.instance_url)
Step 3: Adaptation to a common structure
Palette Learning Platform
Step 1: SOAP_Call(search)
Step 2: [Adaptation to a common structure]

Figure 7 - Search Engine implementation model

e-Logbook and BayFac are both REST-oriented and require an authentication. e-Logbook requires the call of three REST services, and then a manual search on its results (until the search REST service is developed). BayFac requires the call of one REST service to get the classified items instances containing the keywords, and then another call of a REST service for each one found. The PALETTE Learning Platform requires a unique SOAP call to find the results corresponding to the wanted keywords. For each PALETTE service, an adaptation step of the returned results is necessary since each one uses its own vocabulary and structure.

The implementation of this rather simple example has highlighted three main issues:

- Authentication for service access;
- Heterogeneity of vocabularies and structure used in returned information;
- The number of different services calls to the same PALETTE service, required to obtain the requested information.

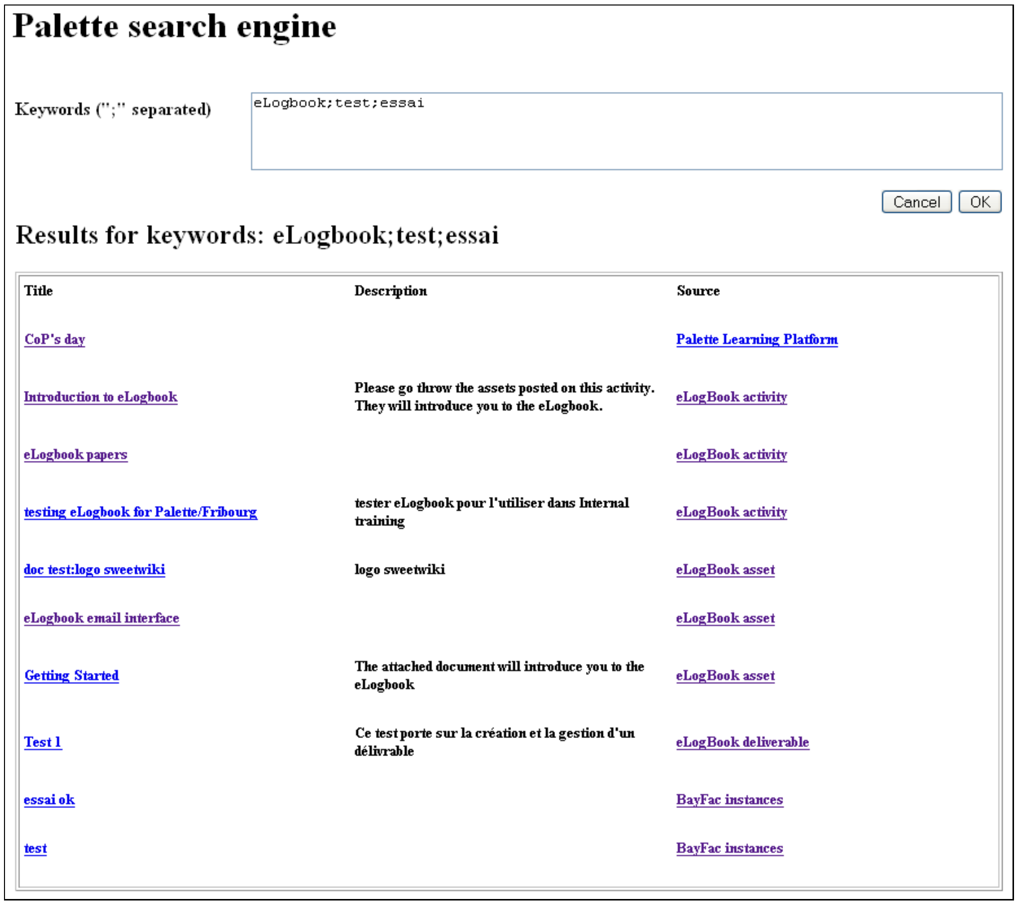
Concerning authentication, two Web services require an authentication, but are not based on the same user database. So each Web service has to be called with different credentials. At the moment, the retained solution consists in making the authentication with test credentials directly in the Web service call, and not via the proposed graphical interface. This problem will disappear when the single sign-on will be available.

The second problem concerns the differences on search result XML formats. The examples below show the different outputs obtained respectively with e-Logbook, BayFac, and the PALETTE Learning Platform.

e-Logbook (assets)	`<assets>`
	`<asset>`
	`<id>137</id>`
	`<name>Soleil Vert</name>`
	`<description>`
	`<p>Voici la liste des participants &agrave; la CoP Soleil Vert </p>…`
	`</description>`
	`<right>reader</right>`
	`</asset>`
	`</assets>`
BayFac	`<instances>`
	`<instance>`
	`http://citi-efficient2/generic/index.php/rest/fs/Document/instance/pinst:i1189084074088402600`
	`</instance>`
	`</instances>`
	`And then:`
	`<instance>`
	`<label>test</label>`
	`<comment/>`
	`<postDate>06/09/07</postDate>`
	`<hasPath>../../uploads/Facette.ppt</hasPath>`
	`<hasURI/>`
	`<classificationState>Classified</classificationState>`
	`<facetVector/>`
	`</instance>`
PALETTE Learning Platform	`<rss version="2.0">`
	`<channel>`
	`<title>PALETTE Learning Platform</title>`
	`<description>The RSS feeds proposes you a list of current PALETTE modules corresponding the search results</description>`
	`<link>http://www.anaxagora.tudor.lu/PALETTE/</link>`
	`<item>`
	`<title>CoP's day</title>`
	`<description></description>`
	`<link>http://lcms.anaxagora.tudor.lu/PALETTE/LCMS/description/detail_28.php </link>`
	`</item>`
	`</channel>`
	`</rss>`

Each result is in XML format, but the schemas are different. We face here a classical interoperability problem, where vocabularies and structure of information need to be adapted to a single common format. In the test implementation, we have chosen RSS, which is a totally ad-hoc choice.

The third problem concerns the number of steps to be realized in order to get the information the composite search service is supposed to provide. Concerning e-Logbook, we first needed to find activities, assets and deliverables, and then realize a full-text search on results. e-Logbook will propose soon a search Web service that will simplify the search realization. BayFac needs two steps to finalize a search: firstly find instances corresponding to keywords, and secondly identify the information for those instances. The PALETTE Learning Platform requires only one step, which is the call of the SOAP service. Yet, it only requires one step because the output format corresponds to the temporary chosen output format (RSS). We face here a granularity problem: search Web services provided by the different PALETTE services do not necessarily return information with the same level of details. Functions of PALETTE services provided as Web services have not the same level of granularity, thus implying that calls to multiple Web services can be necessary on a PALETTE service A while a single call is needed for a PALETTE service B.



Palette search engine

Keywords (";" separated) eLogbook;test;essai

Results for keywords: eLogbook;test;essai

Title	Description	Source
CoP's day		Palette Learning Platform
Introduction to eLogbook	Please go throw the assets posted on this activity. They will introduce you to the eLogbook.	eLogBook activity
eLogbook papers		eLogBook activity
testing eLogbook for Palette/Fribourg	tester eLogbook pour l'utiliser dans Internal training	eLogBook activity
doc test:logo sweetwiki	logo sweetwiki	eLogBook asset
eLogbook email interface		eLogBook asset
Getting Started	The attached document will introduce you to the eLogbook	eLogBook asset
Test 1	Ce test porte sur la création et la gestion d'un délivrable	eLogBook deliverable
essai ok		BayFac instances
test		BayFac instances

Figure 8 - Search Engine prototype

Prototype interface

A prototype interface has been implemented in order to propose a search-engine service accessible by an end-user and provide a simple illustration of the output (Figure 8). Though it is a simple Web page displaying the content of the RSS output, it could have been implemented as a widget to be put in the PALETTE Portal.

Implementation with PI4SOA

In order to realize the implementation in Pi4SOA, we have chosen to build three SOAP Web services to encapsulate the different php functions defined in the direct implementation (see Figure 7). This step was necessary as Pi4SOA is based on WS-CDL, which requires WSDL descriptions of services to call and only manages SOAP services. To actually make the composite service, we had to create choreography in Pi4SOA. This is done in three steps, which we sketch in the following: firstly, the specification of roles and relationships specification, secondly the base types' definition, and finally the choreography flows representation.

Roles and relationships specification

This step enables one to identify the different roles and relationships within the choreography. The relationship will be associated with service calls, and the source and destination roles associated with this relationship. In our case, *e-Logbook*, *BayFac*, and the *PALETTE Learning Platform* are each represented as a role. Then the search engine has two roles, which we named *search engine grouping* and *search engine merging.* This was necessary since a service is called within the search engine: the PALETTE services results merging.

The relationships are:

- e-Logbook searches between the roles *e-Logbook* and *search engine grouping*
- BayFac searches between the roles *BayFac* and *search engine grouping*
- The PALETTE Learning Platform searches between the roles *PALETTE Learning Platform* and *search engine grouping*
- Searching merging between the roles *search engine grouping* and *search engine merging*

Each role has an associated behaviour, which links an interface (generally WSDL) with it. Each participant will have its own WSDL. Knowing that, except for the search engine which is associated with the roles *search engine grouping* and *search engine merging*, all roles are associated with a single participant, we thus have four WSDLs: e-Logbook's, BayFac's, PALETTE Learning Platform's and the search engine's (Figure 9).

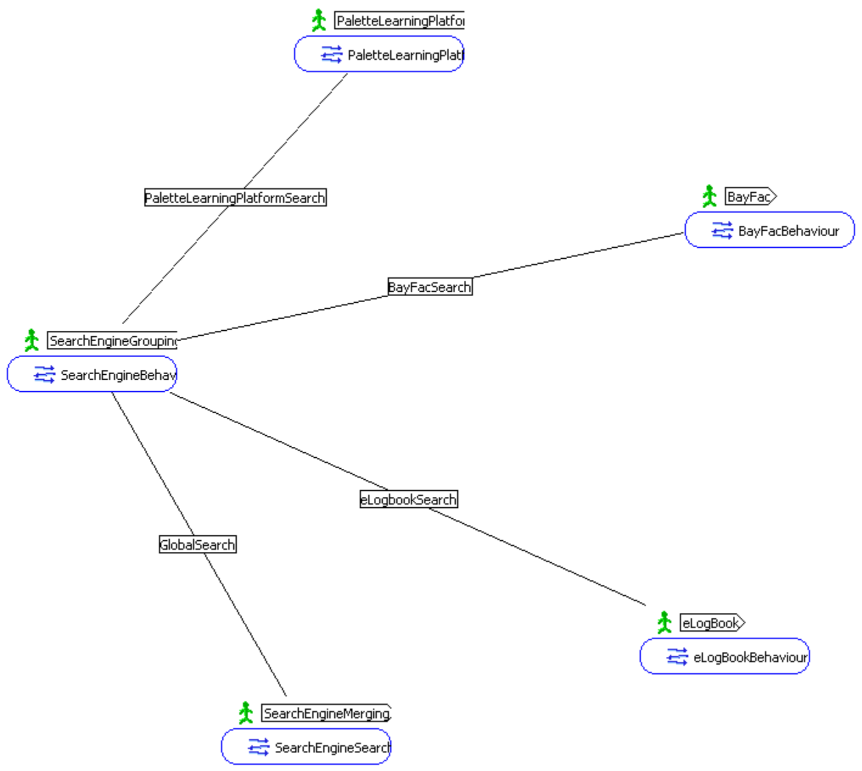


Figure 9 - Pi4SOA - Roles and Relationships

Base types definition

This part, which we do not describe in detail here, reflects the WS-CDL format. Several WS-CDL components need to be specified, such as:

- Name spaces;
- Participant types, representing an organizational unit assuming one or more role types;
- Role types, defined in the roles and relationships part;
- Relationship types, defined also in the roles and relationships part;
- Channel types, communication paths between roles;
- Information types, used to abstract a type definitions for use within other elements of the choreography;

- Token, an alias for a piece of information of a particular type;
- Token locators, a mean of obtaining a piece of information aliased by a token.

Choreography flows representation

This Pi4SOA framework allows us to design the choreography flow. The first step of the choreography specifies that the e-Logbook, BayFac and PALETTE Learning Platform searches will be realized in parallel, followed by the call of the search engine merging service. Each of those elements represents interactions. Let's take as example the e-Logbook search interaction: it will be associated with a channel representing the communication path between e-Logbook and the search engine; this channel will indicate the name of the service to be called (with an exchange element). The exchange element represents the message format, with the input and output of the service. Each interaction is formatted in the same way.

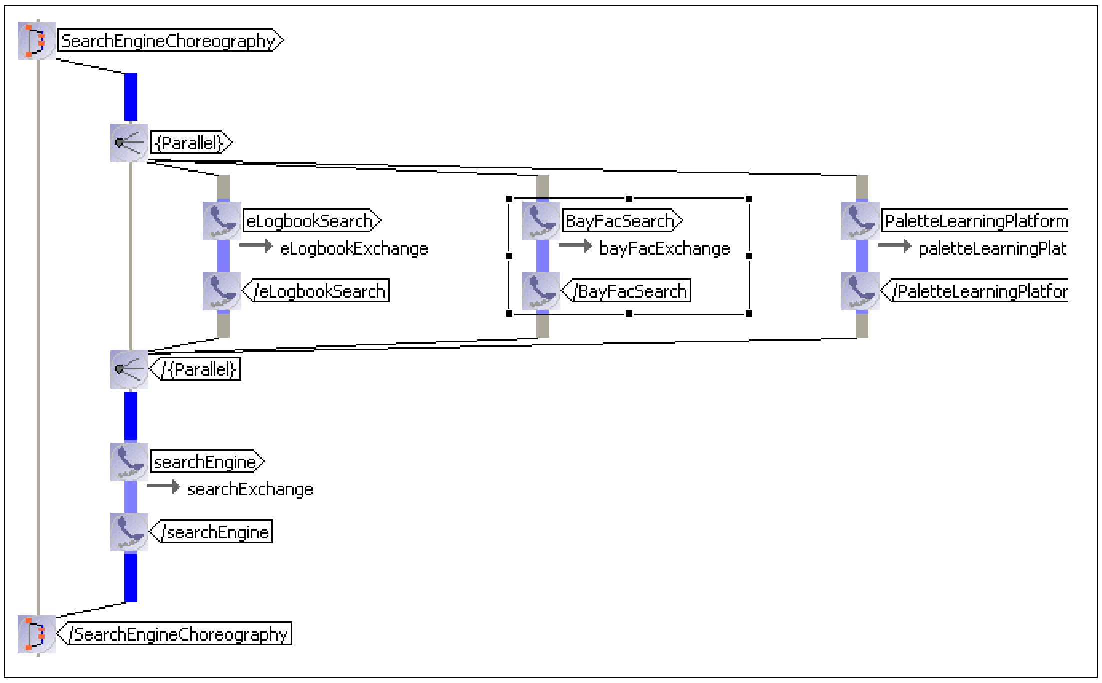


Figure 10 - Pi4SOA – Choreography flows

The first thing we have observed is that the search engine case is not well adapted for an implementation as choreography. Indeed, choreography should represent different participants exchanging information, while in our case we get information from several services and give a result back to the user. Finally, we did not yet succeed in using the WS-CDL execution engine of the PI4SOA framework. Regarding the time spent and the difficulties inherent to the appropriation of WS-CDL, this investigation path has been stopped. Even with comprehensible guidelines explaining how to use Pi4SOA or writing directly a WS-CDL script to implement Web services composition, the developers still will have to understand the underlying concepts of WS-CDL and the way it works. In the current context of PALETTE and the interactions investigated between existing PALETTE services, multi-party conversation, which is the core principle of WS-CDL, is clearly out of the scope.

<u>Conclusions</u>

Starting from the example of a global search tool, which is one of the three identified composite services that CoPs require, we have experienced both manual programming and the use of the PI4SOA framework. Compared to the time spent for manually program the service, it has appeared that using PI4SOA is counter-productive. There are different reasons for that. First, the lack of a proper documentation makes it difficult and time-costly to master. Second, using this framework requires a deep understanding of WS-CDL, as its interface is totally reflecting it. Finally, contrary to what we thought when this framework has been selected; the support for REST services is not achieved. Thus,when SOAP services are not available, additional work is needed because calls have to be encapsulated in SOAP Web services.

We have not tested the implementation of the global search tool with the JOpera framework. Though it would probably have been more adapted as it supports REST calls and allows orchestration-oriented composition, the experience with Pi4SOA and manual programming has shown that - most probably - using a complex framework will not provide benefits, regarding the simple composition needs in PALETTE. Moreover, we face the same problem as with Pi4SOA concerning the lack of proper documentation.

A composition tool could be necessary in the case of a complex composition, which is not the case in the PALETTE context. Indeed, it has been much faster to develop in PHP the global cross-tool search than to try to implement it with Pi4SOA. To conclude, since manual programming seems a simpler and more efficient solution, the alternative proposed in D.IMP.04 with composition at the user interface level with mashups will be adopted. For the next releases of the PALETTE framework, composition will be considered at the later level in the PALETTE portal using the widgets.

3.3.2 Single Sign-On through Portal

Introduction

The PALETTE project consists of a number of PALETTE services. Each service has its individual authentication system. Single Sign-On aims to allow each user to access multiple heterogeneous applications with single identity. Single Sign-On enables each user to login to an application once, and then be able to access a set of applications.

Figure 11 illustrates a generic Single Sign-On protocol. There is an identity provider that consists of a Single Sign-On service and an identity repository. To use Single Sign-On, a user needs to first register to the identity provider for a Single Sign-On identity. For each session afterwards, in the first time a user login to a Web service, he/she needs to request an authentication proxy from the Single Sign-On service for authentication. After the first time, the user can use the authentication proxy to authenticate with Web service directly (thus step 2 can be skipped), and he/she does not need to login by his/her login name and password.

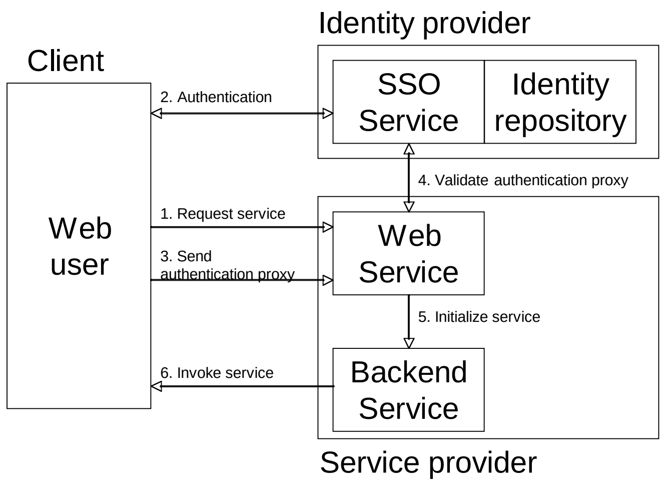


Figure 11 - Generic Single Sign-On protocol.

The benefit of Single Sign-On is obvious: it reduces the troublesome of multiple manually login. Dunne (2003) has discussed the advantages and disadvantages of Single Sign-On:

Advantages

- Improved user productivity: Users are no longer bogged down by multiple logins and they are not required to remember multiple IDs and passwords.

- Improved developer productivity: Single Sign-On provides developers with a common authentication framework. In fact, if the Single Sign-On mechanism is independent, then developers don't have to worry about authentication at all. They can assume that once a request for an application is accompanied by a username, then authentication has already taken place.

- Simplified administration: When applications participate in a Single Sign-On protocol, the administration burden of managing user accounts is simplified. The degree of simplification depends on the applications since Single Sign-On only deals with authentication. So, applications may still require user-specific attributes (such as access privileges) to be set up.

Disadvantages

- Unattended desktop: Implementing Single Sign-On reduces some security risks, but increases others. For example, a malicious user could gain access to a user's resources if the user walks away from his/her machine and leaves it logged in. Although this is a problem with security in general, it is worse with Single Sign-On because all authorized resources are compromised. At least with multiple logons, the user may only be logged into one system at the time and so only one resource is compromised.

- Single point of attack: With Single Sign-On, a single, central authentication service is used by all applications. This is an attractive target for hackers who may decide to carry out a denial of service attack.

Single Sign-On solutions

Today, the two most popular Single Sign-On solutions are OpenID and Shibboleth.

OpenID is an open, decentralized, free framework for user-centric digital identity. OpenID takes advantage of already existing internet technology (URI, HTTP, SSL, Diffie-Hellman) and realizes that people are already creating identities for themselves whether it be at their blog, photostream, profile page, etc. With OpenID, one can easily transform one of these existing URIs into an account which can be used at sites which support OpenID logins. (Quoted from OpenID official Website [Ref:OpenID])

Shibboleth is standards-based, open source middleware software which provides Web Single Sign-On across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner. (Quoted from Shibboleth official Website [Ref:Shibboleth])

The major difference between them is that OpenID still requires users to login to every Websites while Shibboleth does not. Another difference is that OpenID is a pure identity management framework while Shibboleth is an attribute-based framework. Therefore, OpenID allows simpler deployment, while Shibboleth can support authorization through attributes.

Related technology: OAuth

The OAuth protocol enables Websites or applications (Consumers) to access Protected Resources from a Web service (Service Provider) via an API, without requiring Users to disclose their Service Provider credentials to the Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication. An example use case is allowing printing service printer.example.com (the Consumer), to access private photos stored on photos.example.net (the Service Provider) without requiring Users to provide their photos.example.net credentials to printer.example.com. (Quoted from OAuth specifications [Ref:OAuth])

Simply speaking, OAuth is an open protocol for service-to-service authentication and authorization. It enables users to delegate applications to access some protected resources stored in other Web services. Many application portals provide similar functionalities by themselves. Google's AuthSub and Yahoo's BBAuth are two examples. OAuth is not a Single Sign-On solution. It is also not an alternative to Single Sign-On. In fact, OAuth and Single Sign-On are complimentary. We take combination of OpenID and OAuth as an example. With OpenID, users only need to remember single credential when they use OAuth to delegate access rights to applications. On the other hand, with OAuth, users have more opportunities to use their OpenID credentials. This example can also be applicable on PALETTE project.

Single Sign-On solutions for PALETTE project

We will look into two approaches of Single Sign-On solutions for PALETTE project. The first option is to deploy the OpenID framework; the second option is to build a centralized authentication framework on the PALETTE portal.

Approach 1: Deploy OpenID authentication framework

The popularity of OpenID grows very fast in recent years. Nowadays there are many OpenID-enabled Websites [Ref:OpenID_Dir]. Having an OpenID, a user can login to any OpenID-enabled site with single username and password. The easy deployment of OpenID is probably an important factor to its popularity. There are many public OpenID providers available [Ref:OpenID_IdP]. To become OpenID-enabled, Web applications can simply install OpenID consumer libraries to deploy authentication services from the OpenID providers.

The feasibility of OpenID authentication framework is proved through its successful deployment by numerous Web applications. Since PALETTE services are all Web applications, OpenID is also a feasible Single Sign-On solution for PALETTE project.

Figure 12 illustrate an overview of OpenID protocol [Ref:OpenID_Spec]. First, the end user initiates authentication by presenting a user-supplied identifier to the Web application (OpenID consumer) via the user's user agent. Second, the OpenID consumer performs discovery on the identifier for destination OpenID provider. Third, the OpenID consumer and the provider establish a shared secret using Diffle-Hellman Key Exchange for message verification. Fourth, the consumer redirects the user to the provider. Fifth, the user's user agent sends authentication request to the OpenID provider. Sixth, the OpenID provider redirects the user back to the OpenID consumer with either an assertion that "authentication is approved" or a message that "authentication failed". Finally, the user's user agent sends the result to the OpenID consumer, and the consumer verifies the information.

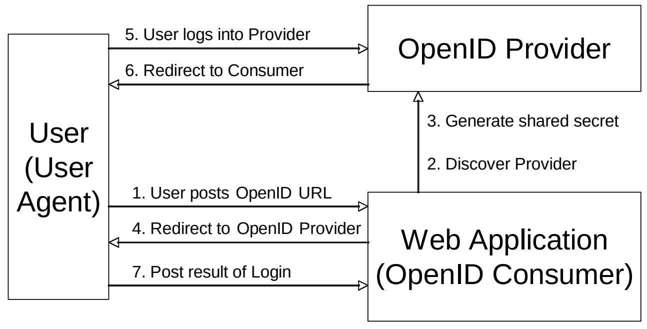


Figure 12 - Overview of OpenID protocol.

Requirements of deployment of OpenID

In the PALETTE project, to deploy the OpenID authentication framework, the PALETTE services will be OpenID consumers. In order to maintain the control of registration of membership of users in the PALETTE project, PALETTE project can choose to build its own OpenID provider. The followings are the requirements of deployment of OpenID [Ref:OpenID_Recipe] in PALETTE project.

Portal needs to build:

- An OpenID identity server (OpenID provider)

PALETTE services need to build:

- A "new database table" to map OpenIDs to internal user IDs.
- A small bit of OpenID UI on "registration page" for new users.
- A small bit of OpenID UI on "sign-in page" for existing users.
- Program codes to support login through OpenID.

Procedure of deployment

How to build an OpenID identity server
There are many software packages for OpenID identity server implementation. They are listed in [Ref:OpenID_Server]. The packages are coded in different languages, e.g., PHP, Java, and Python. In addition, some packages provide other functionalities other than OpenID authentication. Select a suitable software package; then follow its guidelines for installation and configuration. Finally test the server through OpenID-enabled Websites.

How the PALETTE services should deploy OpenID
The following is a brief implementation procedure based on [Ref:OpenID_Recipe].

1. Install OpenID consumer library.
   There are many OpenID consumer libraries available. They are listed in [Ref:OpenID_Lib]. The libraries are also coded in different languages. Select a suitable library; then follow its guidelines for installation and configuration.
2. Create a new OpenID database table.
   This table will be a global registry to look up users by OpenID. It's a many-to-one relationship. Each user can have multiple OpenIDs attached to their account, but a given OpenID can only be claimed by a single user.
3. Add OpenID UI to "registration page".
   Add a section to registration page where OpenID users can sign up using their OpenID. The UI goal should be that OpenID users can easily identify that the site supports OpenID, but that users without OpenID can continue to register normally without being confused.
4. Add OpenID UI to "sign-in page".
   Add a section to sign-in page where OpenID users can sign in using their OpenID. This will work both for existing users of your site that have attached an OpenID to their account and new users, who will be able to sign up using their OpenID.
5. Create a new OpenID login Web page / CGI
   The login Web page needs to:
   i. Look up whether the OpenID entered already belongs to an existing user on the site,
   ii. Redirect to the OpenID provider (using an OpenID library) so the user can authenticate,
   iii. Handle the response from the user's OpenID provider,
   iv. Verify the response (using an OpenID library).

Approach 2: Build centralized authentication framework on Portal

Although OpenID is simple and feasible as a Single Sign-On solution, we may want to have a tailor-made Single Sign-On solution for PALETTE project. For example, a tailor-made centralized authentication framework may integrate authentication management, profile management, authorization management, and API authentication. Figure 13 illustrates the example.

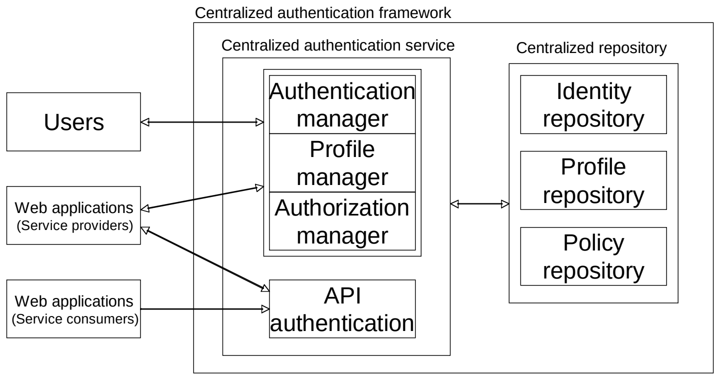


Figure 13 - Centralized authentication framework

This approach has several advantages. First, since the users' information is placed in a centralized repository, the problem of data portability between Web applications is erased. Second, since information of users from different Web applications is placed together, it is easier to support social interaction functionalities for the community. Third, since the authentication, profile, and authorization management services are integrated, the authentication framework can support fine-grained access control.

On the other hand, this approach has several disadvantages. First, the centralized repository becomes the single point of attack. Second, flexibility of centralized framework is low. Third, a tailor-made framework requires more development effort to build its own software package.

Requirements of building a centralized authentication framework on PALETTE Portal

To build a centralized authentication framework to enable Single Sign-On, the PALETTE Portal and PALETTE services need to fulfill the following basic requirements.

Portal needs to build:

- Authentication management service
- This service is used to support authentication for users to access PALETTE services. Popular authentication mechanisms include Kerberos, SAML, and HTTP Digest.
- Identity repository
- Identity repository is the database for storage, management, and query of identity data. An identity repository usually is a LDAP database, or a rational database.
- API authentication (optional)
- API authentication allows users to delegate a PALETTE service to access protected data stored on another PALETTE service. OAuth is an open standard for this purpose. This service is optional because it is not directly related to Single Sign-On.

PALETTE services need to build

- A "new database table" to map "centralized identities" to internal user IDs.
- A small bit of "centralized authentication" UI on "registration page" for new users.
- A small bit of "centralized authentication" UI on "sign-in page" for existing users.

- Program codes to support login through “centralized authentication”.

<u>Summary</u>

Single Sign-On aims to allow users to login to a system only one time, such that they do not need to login again to use the tools in the system. It is a technique that provides convenience to users. In PALETTE project, the PALETTE services are run in different security domains. By deployment of Single Sign-On, PALETTE users can access to the PALETTE services as if they are in the same domain.

OpenID is a simple and popular Single Sign-On solution. It enables users to login to OpenID-enabled Websites with single credential, although the users still need to login to each Website. OAuth is an open protocol aimed for users to delegate applications to access protected data stored in other Web applications. It is a technique that is useful for authentication between applications. OpenID and OAuth can be deployed independently.

To deploy Single Sign-On through Portal in PALETTE project, we have looked into two approaches. The first approach is to deploy OpenID. This approach is simple and fast as the feasibility of OpenID is proved by its deployment by numerous Websites. The second approach is to build a tailor-made centralized authentication framework. Although more effort is required for design and development, it is possible for this approach to enhance Single Sign-On through integration of authorization and profile management. Considering the schedule of PALETTE project, the OpenID approach is the most preferable one.

3.3.3 Common Resource Repository (CRR)

<u>Introduction</u>

In several PALETTE services (such as CoPe_it!, DocReuse, e-LogBook and SweetWiki), users have the possibility to import some resources (images, documents) that will be either consumed to create new resources (DocReuse), or that will be referenced or embedded into new resources (all the others). To implement this kind of functionality, all the aforementioned PALETTE services have developed their internal repository where they put the imported resources or a copy of them.

A basic integration scenario is to support the ability for PALETTE services to archive resources and make them retrievable from within various PALETTE services. This implies that resources must be able to be reused (a) within a particular Service, and (b) across PALETTE Services. In that case, a user will have to upload a resource in a “Common Resource Repository” (CRR) service. Then, each time s/he needs it in the same or in another PALETTE Service, s/he could have the opportunity to find it from the CRR. This is called the “single store scenario”.

<u>Single Store Solution</u>

Single store solution can be applied by following a distributed architecture for p2p repository systems or a centralized one. While the former has the advantages of the maximum up-time operation period, minimized possibilities of crashing, high performance etc, the later is characterized by easiness of development and use. In the context of PALETTE this characteristic seems crucial since the diversity of the existing methodologies and tools has already lead us to a high level of complication from both interoperation and integration perspectives. Furthermore a variety of centralized repositories of resources such as rapidshare [http://rapidshare.com/] and bscw server [https://bscw.ercim.org/] already operate and have gain high acceptance in the Web area.

Following the centralized architecture, there are three important points that have to be addressed for the implementation of the single store functionality:

- The definition of the resource schema.
- The development of the Common Resource Repository.
- The integration of the CRR client mechanism with the existing PALETTE services in order to achieve native support for storing and retreating of resources.

The resource schema

Apart from the content of each resource it necessary for the server to allow a set of extra attributes to be stored as well. These attributes may carry out information about the resource unique id, a resource unique name, the mime-type, the creator PALETTE service, the submission and modification dates, etc. Thus, it will possible for each client to get informed about the resource details before import it to the requestor's service or application. The design of a complete schema for the resources within the context of PALETTE is one of the future goals of WP5.

The Common Resource Repository

The Common Resource Repository will be responsible for storing resources into an internal storage space and for retrieving them. This service will listens to REST or SOAP formed requests from the requestors. Furthermore, some resource management methods for administrating purposes will allow the repository administrator to monitor the status of the repository and maintenance the whole service.

A critical specification that the repository should support is the referencing and addressing mechanism for resources. The referencing mechanism allows transparent access to any resource by simply specifying either the globally unique identification of the resource or the resource's name. The difference between resource identification and resource name is that while the former is automatically generated, the latter is user-chosen (it is provided by the user during the storage of resources). In addition, for each stored resource in the storage place, the service may provide a unique URL from which requestors can download the resource or the resource's content file through a simple http get request. This can be used aiming at providing addressing capabilities to requestors.

Modifications on client-side

In order to integrate the single store functionality with some PALETTE services, it is necessary for these PALETTE services to natively support the invocation of both "store" and "retrieve" methods to and from the CRR. This will require some modifications on these PALETTE services and perhaps the addition of new save and load options in their visual user environment.

Implementation Example: CoPe_it! Object Repository

As described in D.MED.09, CoPe_it! will provide an object repository for storing and retrieving resources together with a set of resources' description metadata. The repository will support all popular content-types from text, pdf and MS Word to images and video. In this service, both addressing and referencing mechanisms will be included. The next step for the application of the particular service is to customize some PALETTE services and try to store and retrieve files or other content to and from CoPe_it! Object Repository. The redefinition of the resource description metadata also plays an important role. However, the choice for the particular storage service has not yet made and it is possible to use one or more alternatives after a collaborative decision.

Issues for further investigation

The existence of a common place where everybody can put and get blobs of information may highlight interesting issues for discussion between the participants:

1. *Authentication and authorization*: many of the free Web storage servers are offering the content without looking for credentials. In these cases it is important for the requestor to know the unique id or name of the resource, or to the entire URL that points to that resource. However, many servers provide secure access to the user's resources and requires user authentication before storing or retrieving resources. In the context of PALETTE where

common repository may play a shared space’s role both techniques can be acceptable, but the first one seems quick and simple.

2. *Browsing and navigation capabilities inside the repository*: This functionality is usually coming together with the first issue. The free of authentication services are based on the existence of a unique address for each resource, and therefore they do not support this kind of functionality. In this way, it is not possible to invoke operations such as “open repository” or “view contents of…’. On the contrary, servers that support access and identity mechanisms can also create storing structures with folders and subfolders and can allow views of specific groups of resources. Nevertheless, some hybrid systems with partially support for both access control and resource browsing are under consideration.
3. *Implementation of a service UI in the context of PALETTE*: Although PALETTE services that want to use the particular service may proceed to the addition of the native support of the service, some other PALETTE services may not integrate this functionality. Furthermore, users that are not using any of these PALETTE services may want to have access to one or more resources into the repository. This, points out the possibility of the development of a Web based UI for the service that can serve requests for storing or retrieving resources direct from users by a simple and user friendly Web form.

3.3.4 Notification of Events

This fundamental service provides the ability to the CoPs members to be notified when a list of selected actions are taking place within the PALETTE services. The entire set of selected actions for notification is depending on the needs of each CoP and will be defined in the next steps of the project. The notification procedure will be mainly based on the cooperation between PALETTE services and the Cross Awareness Knowledge Base (CAKB). A more detailed description of this service will be provided in D.IMP.06 (due in M26).

3.4 Future work

The remaining work towards technical integration aims at the completion of the development of the software components as well as the appliance of the entire architecture by both developers and CoP members. The most critical issues that will be addressed in the next development steps are presented below:

- Completion of the development of all fundamental components of the PSP (Service Registry, Service Composition Engine, Service Delivery Framework);
- Development of the required software components for the implementation of the four support/fundamental services;
- Implementation and application of the four support/fundamental services;
- Implementation and application of the specific scenarios that have been derived from the study of the conceptual integration;
- Focus on support of new scenarios that will be emerged in the future.

4 Conclusions

This deliverable presents the key elements of the integration process in PALETTE. Integration takes place at several dimensions that complement each another: the technical level, the organisational (or activity) level, the users' level. Value is created when there is a seamless process enabling users to perform their activities, improve their practice, and manage their information and knowledge in the smoothest way in order to avoid adding the cognitive burden of technology use above their current work. Then technology becomes a true enabler. Part of the value also comes from the possibility for users to choose the relevant services for them, and then rely on mutualised features that both simplify the general architecture and increase the global performance of the integrated technological system.

PALETTE is on its way to climb a further step. The generic scenarios enable to gather and describe precisely the main streams along which to develop, implement and put in use the innovative services and a creative architecture. They will also illustrate and sustain the way that CoPs could start making a full use of PALETTE findings (technical and organisational) in order to enhance their own practices.

Aiming at supporting the generic scenarios, technical integration is required in order to assure the provision of integrated functionality to both PALETTE developers and CoPs members. As a first step, developers try to establish common means of information representation and interchanging, service creation, publication and interoperation and PALETTE services visualization. Next, they proceed towards the implementation of the four support/fundamental services that have been pointed out as a result of the requirement analysis by the generic scenarios. Future work on this task will enable the implementation of all the highlighted scenarios and will allow the supporting of new scenarios that may be emerged in the future.

5 References

5.1 Webography

All the Web references have been accessed in February 2007.

DPP : DataPortability.org Public Group, at http://groups.google.com/group/dataportability-public

MED09 : Technical specifications of collaboration support tools as Web services, at http://copeit.cti.gr/site/files/D.MED.09_Final.pdf

OAuth: OAuth, at http://oauth.net/core/1.0/

OpenID: OpenID, at http://openid.net/

OpenID_Dir: OpenID Site Directory, at https://www.myopenid.com/directory/

OpenID_IdP: OpenID Providers, at http://wiki.openid.net/OpenIDServers/

OpenID_Lib: OpenID libraries, at http://wiki.openid.net//Libraries/

OpenID_Recipe: A Recipe for OpenID-Enabling Your Site, at http://www.plaxo.com/api/openid_recipe

OpenID_Spec: OpenID Authentication 2.0 Specifications. http://openid.net/specs/openid-authentication-2_0.html

OpenID_Server: Run your own identity server, at http://wiki.openid.net/Run_your_own_identity_server/

OSG : OpenSocial API group on Google groups, at http://code.google.com/apis/opensocial/

PIS : Pi4SOA project wiki, at http://pi4soa.wiki.sourceforge.net/

Shibboleth: Shibboleth Project, at http://shibboleth.internet2.edu/

5.2 Bibliography

Chris Dunne. “Build and implement a single sign-on solution”. IBM developerWorks, 2003. Available in http://www.ibm/developerworks/web/library/wa-singlesign/

Elisa Bertino, Elena Ferrari, "XML and Data Integration," *IEEE Internet Computing*, vol. 05, no. 6, pp. 75-76, Nov/Dec, (2001)

Engeström, Y, (1987) Learning by expanding

Engeström, Y. (1999); Engeström , Y., Miettinen, R., Punamäki, R.-L.,(Editors) Perspectives on Activity Theory (Learning in Doing: Social, Cognitive and Computational Perspectives), Cambridge University Press, 1999.

Porter (1985), Competitive Advantage: Creating and. Sustaining superior Performance, Free Press, 1985.

Wenger, Etienne, Communities of Practice: Learning, Meaning and Identity, Cambridge University Press, Cambridge, UK, 1998.

Wenger, E;, McDermott, R, Snyder, W.M., Cultivating Communities of Practice, Harvard Business School Press, 2002.

APPENDIX A

Detailed Implementation Descriptions of ad hoc scenarios

e-Logbook ↔ CoPe_it!: Transparent profile synchronization

Knowing that CoPe_it! and e-Logbook are stand-alone Web applications independent from each another, it would not be an optimal solution to centralize profile information in one place. Nevertheless, users should still be allowed to choose to automatically or occasionally synchronize common profile data stored in the different Palette tools, as this would save them from the burden of repeating the same actions in two different places.

It is also worth noting that there are some application-dependent profile data, needless to be propagated from one tool to another. As an example, the option «public profile» in e-Logbook could be set to true or false, depending on whether or not the user wishes to make their account visible to all other e-Logbook registered users. This option does not exist in CoPe_it!, and so, there is no point in propagating any corresponding change. As for common profile data, such as home address, phone number, picture, and Website, an agreement between tools to follow the same naming conventions makes it easy for users to identify these common fields as he/she visits the different Web applications, and makes it easy for services to communicate the corresponding modifications (the presentation or visual integration level is concerned with this issue).

The implementation of this transparent profile synchronization feature can be described as follows. In the e-Logbook page where users can fill in their profile information, the options to always or occasionally propagate changes related to common profile data, to CoPe_it are made available. Then, every time the user profile information is updated in e-Logbook, the latter checks whether any of the two options mentioned previously are set to true. If this is the case, then e-Logbook invokes an external rest Web service responsible of updating the user profile data in CoPe_it!.

Using an OpenID solution with API authentication will help make this process transparent to the user, as he/she will not be requested to enter login information for CoPe_it. As a matter of fact, he/she will only need to authorize e-Logbook to invoke the external service on his/her behalf. After having the user's delegation, CoPe_it! will generate an access token for e-Logbook to access the needed service. e-Logbook can then use this token to invoke the function responsible for updating the user profile data in CoPe_it!. The same process is followed to propagate profile information from CoPe_it! to e-Logbook. Finally, the same procedure can also be followed to propagate profile information changes among other Palette tools.

SweetWiki ↔ CoPe_it!

The overall goal of this integration effort is to augment existing services of CoPe_it! by utilizing the ontologies (and basic KM services) that have been developed in the context of Palette. In particular, future releases of CoPe_it! will permit the annotation of individual resources found in collaboration spaces by CoP members (as well as annotation of entire workspaces), in order to explicate the role these resources play within the problem domain. The proposed mechanism is that of tagging, i.e. the ability to add tags to any resource, that are characteristic for the domain of the issue being discussed. This will make the CoPe_it! system aware of the semantics of each resource. By having the machine being aware on the semantics, some provided services can be elevated to the semantic and domain specific level. More precisely:

- *Enable semantic search of resources and collaboration patterns*. Currently, search services in CoPe_it! operate on a simple keyword basis. This means that search services can only match exact keywords against some attributes of resources. While this is useful in some cases, it cannot answer queries that require information on a semantic level such as "Find all evaluation reports that contain the term X" or more complex queries such as "Find all research papers that have been refuted by empirical evaluation". The semantic search also make it possible to found resources that are not explicitly sited in a workspace, e.g. if a workspace contains a PhD thesis containing the word X, it can be retrieved as a possible result for a query such as "find all the research papers containing X" since a "PhD thesis" is a sub-class of "research paper", this example shows also the possibility to combine (a *priori*, by reducing the search space, or a *posteriori* by filtering the results) semantic search with keyword-search rather than replace it.

- *Facilitate the understanding of a collaboration space within the context of awareness services and aid the formalization process*. By tagging individual resources users will be able to understand better the role individual resources play in a collaboration space. Moreover, the tags may also be used as one additional parameter during the process of formalizing the collaboration space, as supported by CoPe_it!

The ontologies created with the Palette KM tools can play a catalytic role in with respect to tagging resources in CoPe_it!. In particular, they permit establishing a common vocabulary that describes precisely and sufficiently a domain. Tags for annotating individual resources in CoPe_it! collaboration workspaces can then be chosen from the ontologies related to the appropriate domain. As a result, the semantics of resources can be machine-processable and hence the aforementioned CoPe_it! services can be elevated to the semantic level. The integration of semantic services into CoPe_it! aims at supporting this resource tagging functionality.

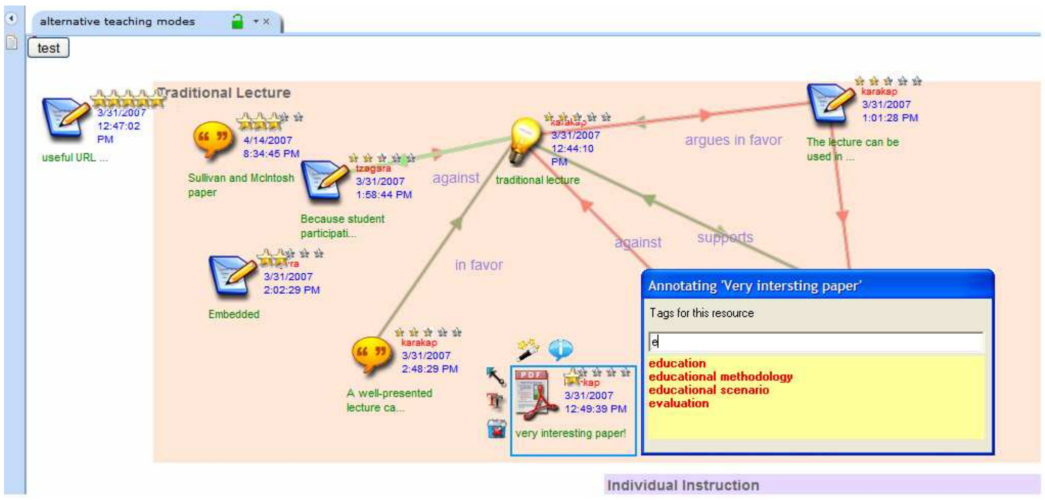


Figure 14 - The dialog box enabling users to annotate the selected resource. While the user types in the tag(s), the system auto-completes the entered text by suggesting concepts from the ontologies that match the text.

Figure 14 illustrates what the user will be able to do once the integration with KM has been completed. Users will be able to tag any resource (document, idea, comment, relation, aggregation mechanisms and even entire workspaces) found in a collaboration space with concepts from an ontology that has been authored using Palette's KM tools (or any existing ontology in the Web, for some domains, specific ontologies exist and can be used by CoPs). Upon selecting an item, a special option will be visible through which users may add, edit and remove tags to the selected resource. By selecting the add option, a dialog box will appear enabling the user to type in the tag(s) that he wishes to assign to the resource. This mechanism can easily be extended to a set of resources or to the collaboration workspace. The dialog box will auto-complete the text entered by the user: i.e. while the user types in the text, CoPe_it! will propose labels from the ontology that are associated with concepts of the ontology and match the entered text. Hence, the user will be able to select only tags that match labels in the domain ontology. Once the appropriate tag has been selected it is stored into CoPe_it! as part of the resource's metadata.

To achieve the functionality described above, a server-to-server communication between CoPe_it! and basic KM services is required. Figure 15 outlines the sequence of events that take place, while the user types in the text in the dialog box when attempting to tag a selected resource. The foreseen integration will make use of the Web Services as they were described in D.KNO.03.

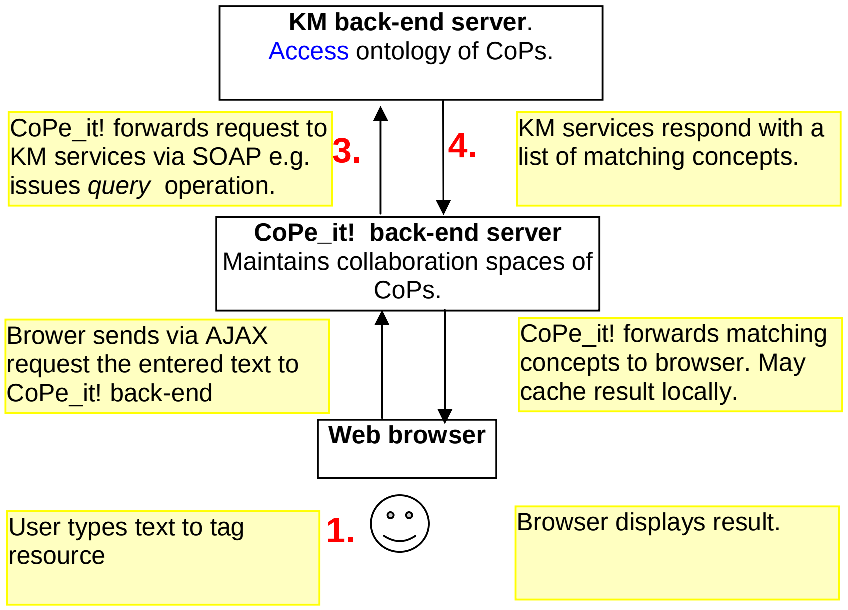

KM back-end server.
Access ontology of CoPs.
CoPe_it! forwards request to KM services via SOAP e.g. issues *query* operation.
3.
4.
KM services respond with a list of matching concepts.
CoPe_it! back-end server
Maintains collaboration spaces of CoPs.
Brower sends via AJAX request the entered text to CoPe_it! back-end
CoPe_it! forwards matching concepts to browser. May cache result locally.
Web browser
User types text to tag resource
1.
Browser displays result.

Figure 15 - Sequence of events to annotate an item in CoPe_it! During this interaction, CoPe_it! back-end contacts KM services back-end using SOAP requests.

The communication between CoPe_it! and KM services will involve the following steps:

- The user enters text to tag a selected resource;
- The browser issues an AJAX request to CoPe_it! requesting completion of the entered text.;
- CoPe_it! back-end forwards request to the semantic server issuing a query operation and supplying a valid SPARQL expression;
- The semantic server sends response in XML to CoPe_it! server;
- CoPe_it! processes received response and transforms it into a format suitable for the Web browser;
- Web browser receives response from CoPe_it! server and displays result to the user;
- Once the user selects the desired tag, it is associated with the resource and stored along with the resource's metadata.

It is important to notice that the described tagging service will also take into consideration the user's desired language and propose tags only in the specified language. Information relevant to a user's language preferences is available in his/her profile.