COLDEX

Collaborative Learning and Distributed Experimentation

Information Society Technologies Programme

Project number: IST-2001-32327

Network Specification

Deliverable Number:	D.6.1.1
Contractual Date of Delivery:	M12
Actual Date of Delivery:	June, 2004
Version:	3.0 (Final)
Work-Package:	6
Lead Partner:	UNED
Contributing partners:	VXU
Authors:	Verdejo, M.F.; Barros, B.; Read, T.; Mayorga, J.I:, Velez, J., Celorrio, C. Y. Calero
Contact:	bbarros@lsi.uned.es

Contents

1 Revision History

Date	Name	Version	Description	Authors
25th February 2003	Network Specification	0.1 (Start of the first design cycle step)	Early draft on the NetSpec document	UNED Group
26th March 2003	Use Cases	--	Use Cases for the network specification in UML	VXU Group
6th May 2003	Use Cases	--	Further comments on the previous Use Cases document	VXU Group
12th May 2003	Network Specification	1.0	Complete set of use cases for the network specification. It regards the VXU and UNED visions and proposes a first step towards a unified document	UNED Group
28th May 2003	Response to the Network Specification v1.0	(Start of the second design cycle step)	Considerations on the Network Specification v1.0	VXU Group
3th June 2003	Comments to the VXU response	--	Answer to the VXU response	UNED Group
October, 2003	The COLDEX Metadata Synchronisation Service (MSS) and other services			UNED Group
18th-19th December, 2003	Metadata workshop in Madrid		Metadata workshops and requirements for loading and searching LO	UNED Group COLLIDE Group VXU group
22th January, 2004	Experiment Ontology		Experiment Ontology and Metadata proposal	UNED Group
4th March, 2004	UDUI comments for the ontology and metadata			COLLIDE Group
6th May, 2004	UDUI examples for CoolModes		Answer to petition of types of LO	COLLIDE Group
15th June 2004	D6.1.1	2.0	Final version of deliverable	UNED Group

Table 1. Revision history of the document

2 Introduction

With respect to its contents and definition, this is a document for D6.1.1 COLDEX deliverable, dealing with the specification of the distributed COLDEX Server Infrastructure. Nevertheless, we are also including some tasks which would have been a part of the D6.2.1 deliverable, which addresses the System Prototype 1, that is, the implementation of the COLDEX server software, including interfaces for people and institutions.

With respect to its management, this document is the result of a design process in several design cycles involving the COLDEX partners (see section 2 for a summary of the management and table 1 for a revision history). Their feedback and criticism has been reflected in the final deliverable documents.

The document is organised as follows: In the next section the Architecture of the COLDEX LOR is described as well as the general architecture of the COLDEX network. Then, the components of the architecture are described and finally, in the section 5, some services offered by the COLDEX Portal are described. An appendix includes the documents used in the analysis and design of the prototypes.

3 Architecture

The architecture of this system can be seen in figure 1 below. It has been the result of several design cycles, which documentation can be read in the appendix. As it is described in section 3.3, the final choice consists in using LDAP as a storage access mechanism and framework. LDAP allows for distribution and replication, which makes it appropriate for Coldex needs. Furthermore, next section presents the design rationale and requirements.

3.1 Architecture of a COLDEX LOR site

The principles informing the architectural design for the Coldex System had to satisfy several important requirements, namely, it had to be distributed, extensible, keep semantic information (collected in a declarative way, for increased ease of use), adaptable and, as much as possible, close to the standards.

The design process was guided by these desirable properties. Its final result can be seen in figure 1 and shows how the architecture is made up of 4 components, the Coldex Portal, the Knowledge Manager, the Social Manager and the Learning Object Repository (LOR):

- **Coldex Portal**: The portal is a web application that enables users to interact with all modules in the Coldex architecture. As such, it offers individual workspaces to users and operates as an interface to working groups and learning objects stored in the LOR. The underlying LOR storage mechanism is a LDAP server. In order to install the portal in a server, Tomcat 5.0 needs to be pre-installed together with Java. The portal is therefore installed as a new web application.

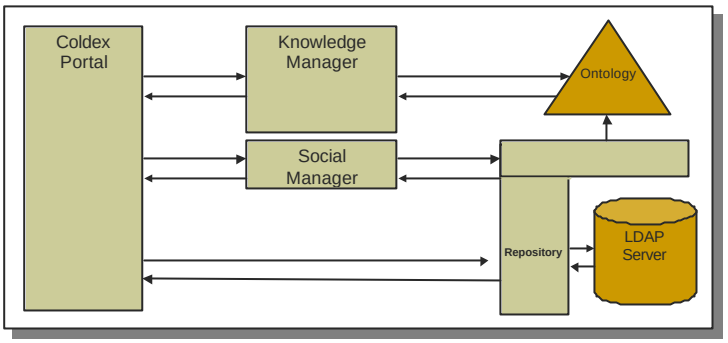


Figure 1. Architecture of the Coldex System

- **Knowledge Manager**: This module allows users to interact with knowledge stored in the protégé ontology, which can be used to separate the system knowledge from the actual code that implements it. Furthermore, this separation minimises the cost of introducing changes into the knowledge in the system and facilitates the personalisation of concepts held in the system.
- **Social Manager**: This module stores all the information related to social aspects such as the users that can connect to the system, the organisation of groups, previously active sessions. Furthermore, it enforces the policies related to user roles within the system and records accounting data about user access, tool usage, etc.
- **Learning Object Repository**: This module stores the LOs and manages their access and physical location within the Coldex server. The low level functionality is achieved by using LDAP.

3.2 Architectural protocols

The MSS was originally conceptualised as a MOM (Message Oriented Middleware) system reflecting the standard application of messaging to the synchronisation of loosely coupled, reliable enterprise frameworks. MSS within COLDEX was defined to be a many-to-many messaging example. The extension of JMS into Message-driven beans (MDBs) EJB components enables messaging to be managed making use of all the benefits of J2EE servers. While a message-driven bean is responsible for processing messages, its container takes care of automatically managing the component's entire environment, including transactions, security, resources, concurrency, and message acknowledgment. Such a view was coherent with the initial conception of the COLDEX server architecture as containing a robust and scalable mechanism for handling the updates of local copies of the metadata catalogue in solid way where the updates are guaranteed even though the COLDEX Server installations are loosely coupled.

Since the initial conceptualisation of the COLDEX server architecture, a lot of work has been undertaken to better define different functionalities including the Learning Object Repository and Portal. Part of this research has highlighted the role that LDAP (Lightweight Directory Access Protocol) and JNDI will have in the overall server architecture. LDAP is a protocol for accessing directory services, specifically X.500-based directory services (which runs over TCP/IP or other connection oriented transfer services). Work with LDAP by the UNED group has shown it to have certain “implicit directory replication” facilities that make it an interesting candidate for the MSS instead of JMS.

3.3 Applying LDAP to COLDEX

As has been documented in appendix (section 8), initially it was contemplated that the MSS should be implemented using JMS and Enterprise Java Beans. However, this option was discounted both for technical reasons and also to make the most of the LDAP server which is being used as the underlying storage mechanism for the LOR (and is detailed in the section 5). The LDAP server (figure 2) consists of a database engine that stores attribute-based information. The server provides naming and directory services to locate resources and is useful for metadata-based searches. It is particularly useful in distributed environments, automatically handling the update of duplicated servers.

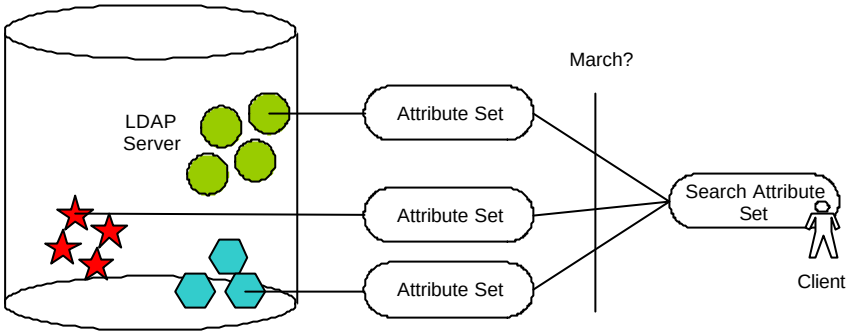


Figure 2. LDAP mechanism in the system

3.4 General architecture for COLDEX network

As can be seen in figure 3 each partner has their own installation of the Coldex system (portal, repository, etc.). As such each repository is physically separate from the rest although they are in fact mirrors of each other. LDAP handles the underlying synchronisation of data, propagating changes from an originating repository to the others. Searchers are therefore undertaken locally preventing problems due to network delays, etc. It should be noted that the system is distributed in the sense that learning objects and projects, etc., are synchronised remotely between partner installations.

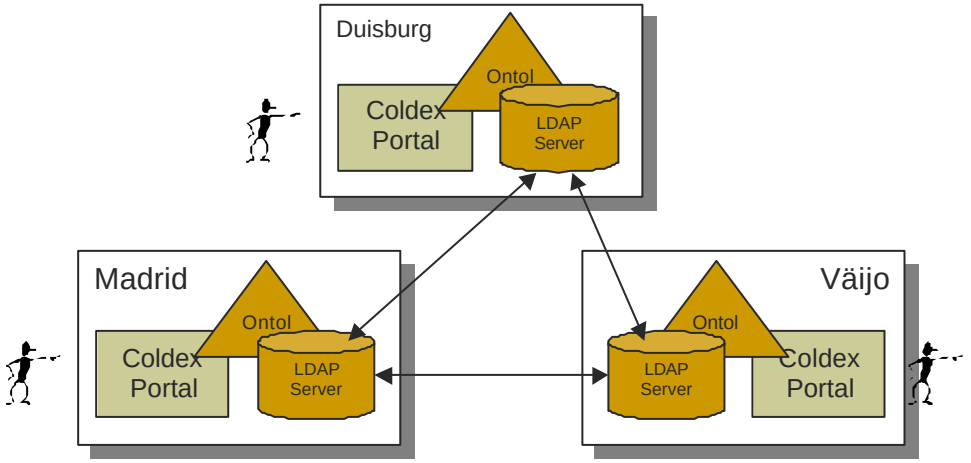


Figure 3. Architecture of the Coldex Virtual Commnity

Each partner system has been designed as a software architect that hides the underlying business logic, offering the client as such, a collection of services that encapsulates its behaviour. Such an architecture is understood as being a collection of components that are connected together to provide an overall Coldex system functionality. The structure of a component in this system can be seen in the following figure 4.

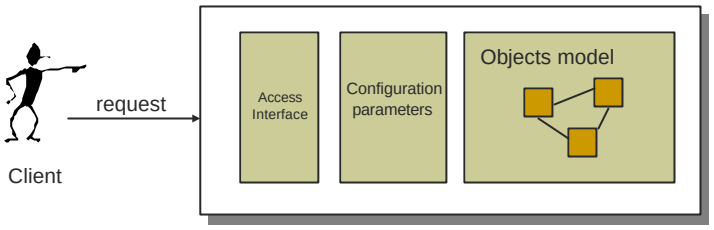


Figure 4. Structure of a component

In order to design and develop an architecture of this type it is necessary to define the components that will be needed, consider their instantiation, and establish the dependencies between them. The architecture resulting from this process can be seen below in figure 5. As should be the case in a system of this type, the design has been separated into the three levels of "viewer", "controller" and "data". The view level presents the data contained in the model to the client. The controller encapsulates the data associating actions to components to construct the data presented to the client. Finally, the underlying data in the system is contained in two sources, the ontology and the LDAP server, which holds the LO repository.

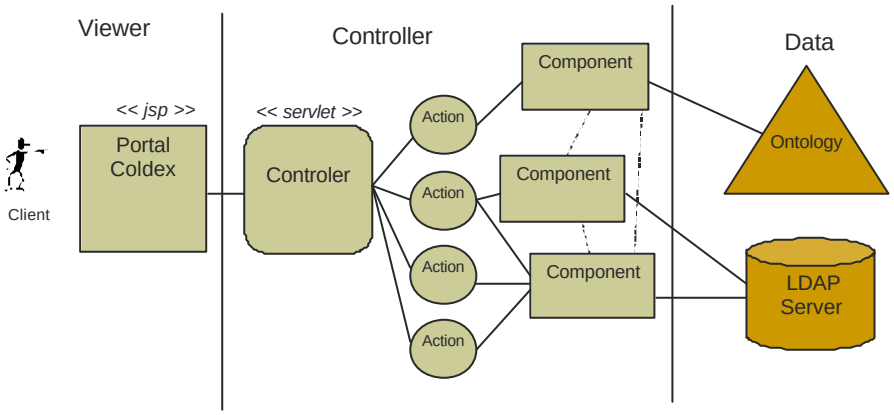


Figure 5. View/Controller/Data architecture for the system

4 Component Description

In this section the components will be detailed.

4.1 Repository Component

The repository contains objects and is responsible for maintaining their containers and associated metadata (learning objects – explained in the section 5.2 where the Learning

Class Component is detailed). Furthermore, it maintains the distributed copies of this information (in the different partner installations) up to date. The objectives that the repository fulfils are:

- LDAP attributes are used to facilitate the search process
- Each LO has its own metadata register
- Each register depends on the LO type
- A register is defined to consist of a collection of attributes
- Searching can be undertaken by filling in fields on a search form based upon the metadata associated with LOs, and the results are returned to the user as LOs

These functionalities constitute a total or partial implementation of the following use cases: 7.2.4.14, 7.2.4.15, 7.2.4.16, 7.2.4.19, 7.2.5.2, 7.2.5.3, 7.2.5.4, 7.2.5.6, 7.2.6.2, 7.2.6.6, 7.2.6.8, 7.2.8.1, 7.2.8.2, 7.2.8.3, 7.2.8.4, 7.2.8.5, 7.2.8.6 in section 7.2.

The structure of the Repository, as can be seen in figure 6, provides an interface to the underlying LOs stored on the LDAP server. It is simply configured by defining certain parameters such as the route to the local directory where files should be stored, the LDAP name context, and who is the owner of the repository. Once configured accesses to the data are abstracted via simple methods such as add, remove, and lookup.

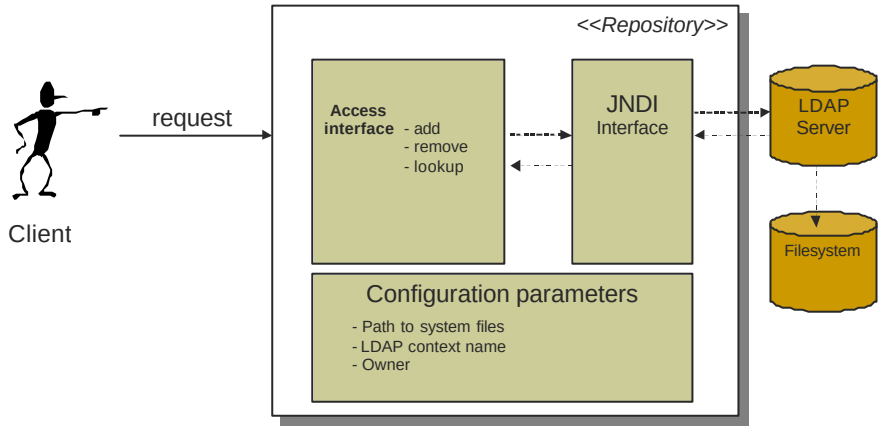


Figure 6. The Repository Component

In the figure 7 the structure of classes which make up the repository can be seen, defining the main methods in each case.

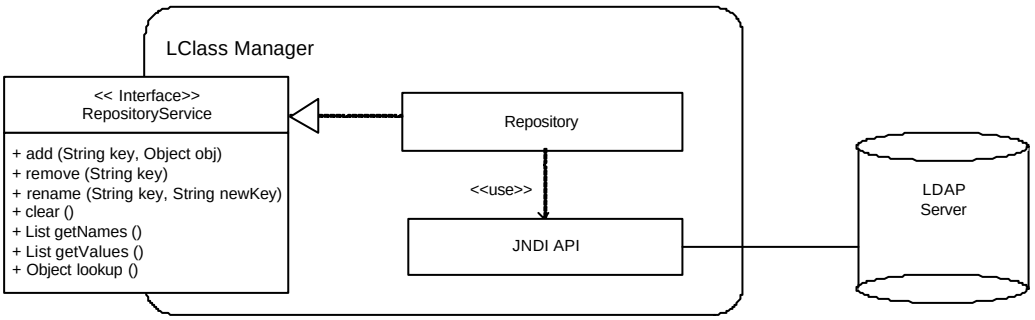


Figure 7. Structure de classes for the Repository Component

4.2 Learning Class Manager Component

A learning Class is a zip package of information about a learning object. To use certain types of learning objects in the system, the user with manager role must install the learning class of this learning object into the system. This process gives rise to certain questions: how to present a learning object to a user? How to create learning objects of this type? Which specification of metadata should be used? How to populate the ontology with metadata? How to auto-populate metadata fields?

To answer these questions we need to consider a learning class in more detail. A learning class can be seen to consist of the following:

- **Views**: the presentation of the LOs is determined by two factors:
  - **Style**: the visual style used for the deployment of the LO (achieved by using CSS)
  - **Layout**: the way in which the LO is related visually (achieved by using XSLT)
- **Metadata schema**: the metadata schema used here are:
  - **Coldex Schema**: a special set of metadata has been developed to meet the needs of this project and goes beyond the standard available metadata.
  - **LOM Schema**: the LOM metadata schema
- **Learning Object Schema**: the schema used here for the LO content:
  - **Schema**: LO validation (DTD)
- **Forms**: details of the forms used to manipulate the LOs depend upon:
  - **Style**: visual appearance of the form (CSS)
  - **Layout**: composition of the form (XSLT)
  - **Xform**: form schema for adding new LO

An example of the way a Learning Class is defined can be seen in figure 8. In this example a zip in which package "Tool.zip" can be seen to be composed of a manifest which contains general information about the LO, the CSS and XSLT templates used to define the way it can be presented, the different metadata for both the LOM and the Coldex Schema, the DTD used to validate the LO structure, the presentation structure used for the forms, and finally, the typical queries used to handle instances of this LO.

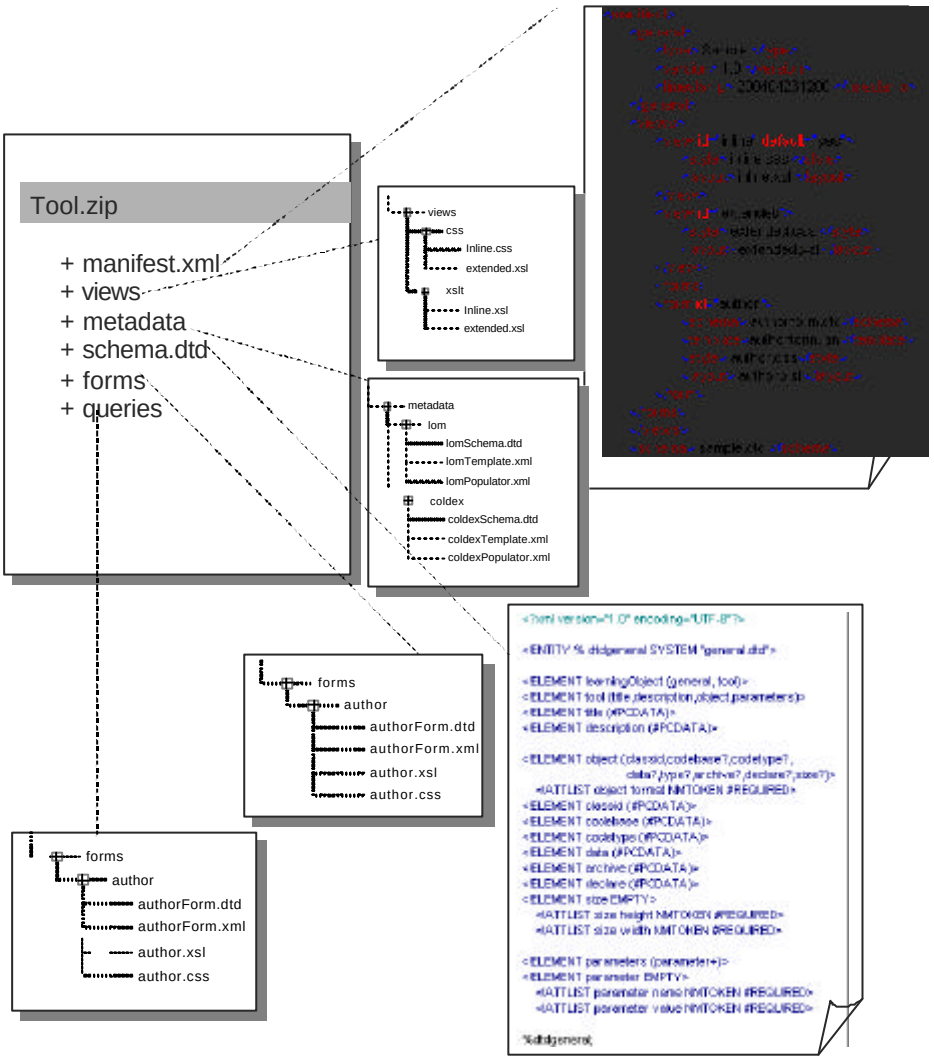


Figure 8. Example of a LO

The implementation details of the Learning Class can be seen in the figure 9, which distinguishes between the LClass manager which makes up the LO interface and encapsulates the different behaviors of the repository, and the actual LearningClassRepository which instantiates the RepositoryService and actually provides the access to the physically stored objects.

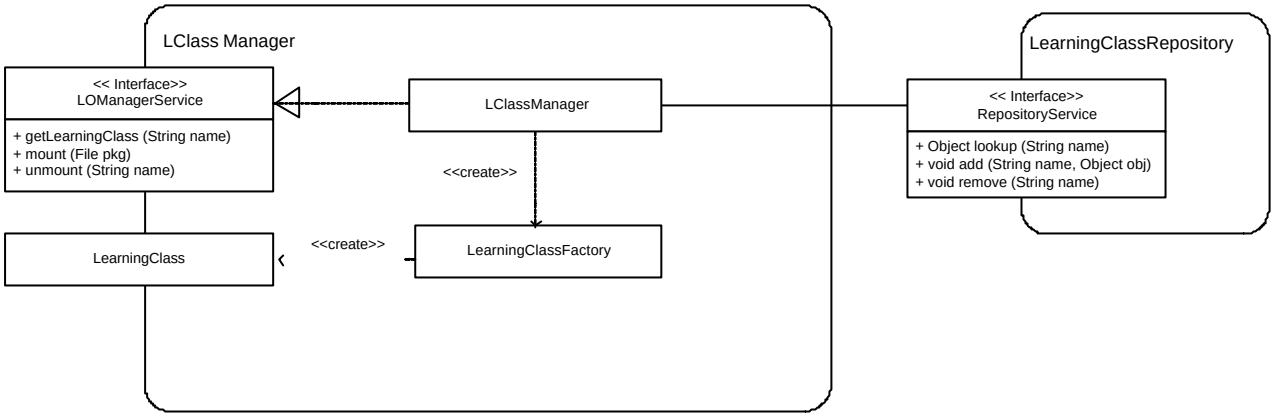


Figure 9. Class diagram for the Lclass Manager

4.3 KM Component

The Knowledge Manager Component (KMC, for short) is entitled to exploit the knowledge within the Coldex Portal. Its main purpose is to take advantage of the ontological model underlying the Portal. The system dynamics-in the sense of services, user interface and customisation-are managed by the Social Manager Component. Hence, its behaviour is adapted to the actual groups using the system.

This component implements the use cases referred in section 4.1 as well as, totally or partially, the following use cases: 7.2.4.1, 7.2.4.2, 7.2.4.3, 7.2.8.7, described in section 7.2

An ontology is an explicit specification of a shared conceptualisation (Gruber, 1993). Using an ontology for describing declaratively a conceptual model provides the KMC with a greater flexibility as a) the code is separated from the model, which, in turn, b) minimises the cost of making changes, c) the functionalities are not "frozen" in the current code and, so, d) fosters the customisation of the concepts within the model.

The KMC is designed to manage this ontological model. This means making use of the complex knowledge retrieval capabilities that the ontology offers to cope with the specific necessities of the Portal users. It encloses the communication with the ontology but takes it apart from the user (the calling program, like an Struts action) isolating the internals of writing complex queries from the software using the KMC. Furthermore, it communicates with a query repository, which collects a variety of useful retrieval possibilities.

Figure 10 shows how the KMC is able to answer the queries sent to the ontology through its *search* service. For doing so, the client (a struts action, in fact) has to select one of the ontological queries stored in the query repository and add a number of metadata for filling in the query parameters.

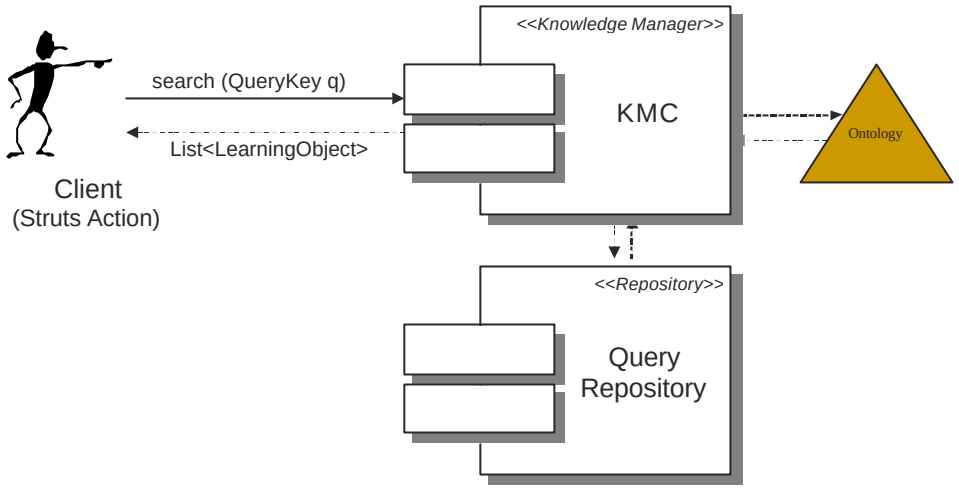


Figure 10. The *search* service: how the KMC answers queries

The KMC offers a public interface, *KnowledgeManagerService*, which defines the *search* method. The class diagram can be seen in figure 11. The KMC incorporates a generic search engine, which specialises into two particular engines, which are able to deal with the queries posed in two available languages[1]. There is also a generic query class with subclasses corresponding to the syntax of the referred languages, which differences can be seen in figure 12[2]. The class diagram also shows that the KMC uses the Query Repository (which stores generic queries as a response to the KMC needs) and the Learning Object Repository.

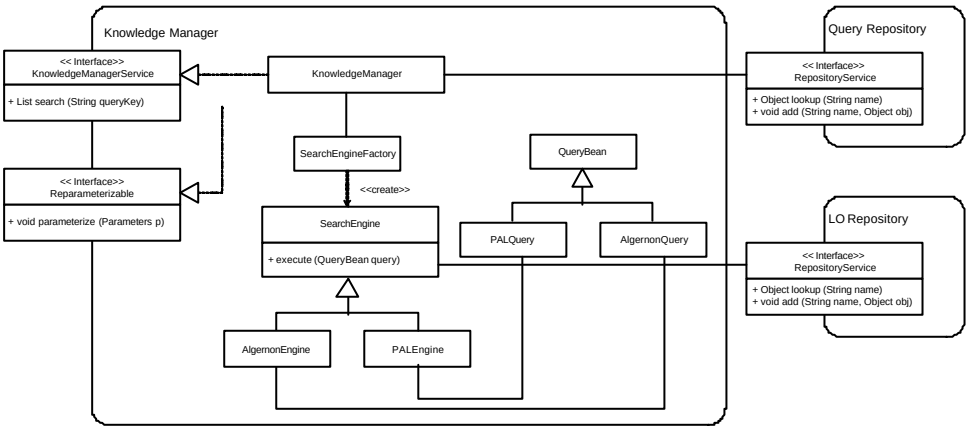


Figure 11. Class diagram for the KMC

[1] We are currently using PAL (Protégé Axiom Language) and Algernon. These languages are frame-access languages, i.e., subsets of First-Order Logic. Both can be used along with Protégé knowledge editor.

[2] A PAL query consists of two parts, a declaration of the variables to bind by the query and the query statement itself. An Algernon query is a succession of *access paths*, being an access path any triplet in the form (*property object value*), where a *property* is, usually, a slot belonging to the *object*.

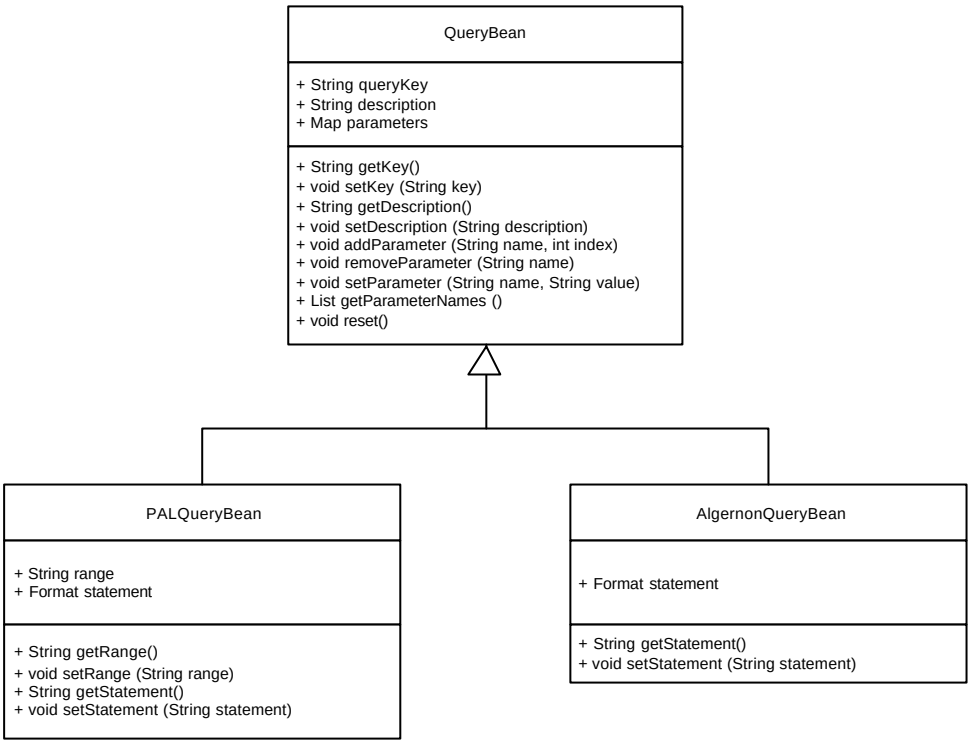


Figure 12. Generic query class and its subclasses for the currently available languages

Figure 13 is a sequence diagram for the *search* use case. The operations go as follows: firstly, the KMC has to be parameterised, which configures its searching capabilities. Then, a query can be created by the user or selected from the query repository. The type of the chosen query will allow instantiating the adequate search engine. Before running the query, its parameters have to be filled in after the values of available metadata. The actual query execution will consist in a dialogue with the Ontology and the Learning Object Repository, which will return a list of Learning Objects as the answer to the query.

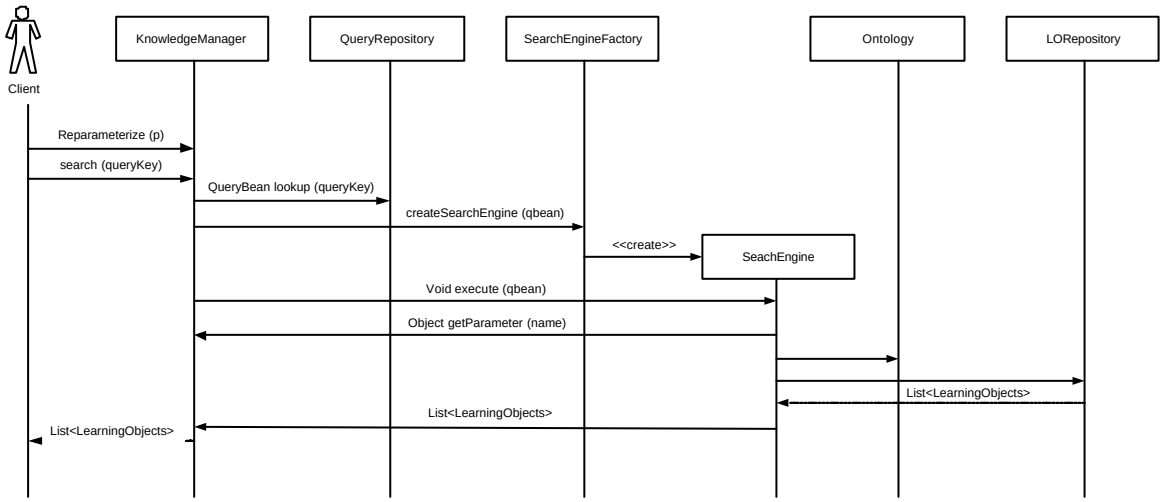


Figure 13. Sequence diagram for the *search* use case

4.3.1 The Ontology

As we have written before, the KMC uses an ontology for describing the system knowledge. The main idea is having the full potential of such a representation artefact in order to enhance the retrieval possibilities already available for the LOR. We have chosen Protégé as our knowledge editor[3] because of its versatility and ease of use.

Figure 14 depicts a rather quick sketch of the Learning Object taxonomy, showing concepts such as *Resource*, *Framework* or *LearningDesign*, which are proposed for helping to declare learning objects in the context of Coldex experimentation.

Figure 14. A sketch of the Learning Object Taxonomy

The ontology *concepts* hold a number of *properties* (often referred as *slots* or *attributes*), which establish relations between instances of these concepts. Those properties are also related to the objects' metadata, which implies some degree of redundancy between the ontology and the LOR.

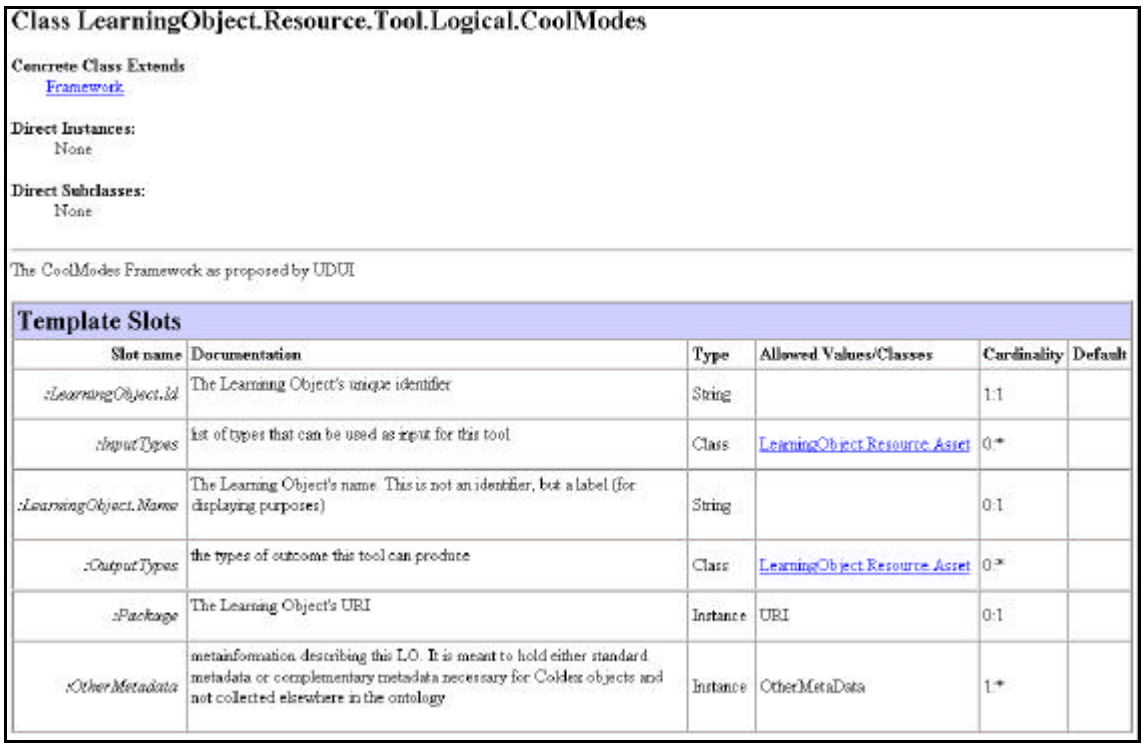
Rather than being a closed model, the ontology is meant to be enlarged-in a declarative way-for coping with new tools and scenarios, which could be thought relevant for Coldex environment.

4.3.2 LO and metada

Working teams may decide their own metadata sets, which can be refined gradually within the system. Proposing a set of metadata means providing a DTD or a template for each LO type. Section 5.2 explains how to define a LO type as well as its corresponding metadata. Figure 15 shows an example of the LO type CoolModes, a kind of Framework, which, in turn, is a sort of Resource.

Besides, figure 15 presents the inner structure of a type of Learning Object, the CoolModes Asset. This includes a number of generic metadata as well as some specific ones meant to serve Coldex requirements.

[3] We are using Protégé 2.0.1, build 168. Furthermore, we are working with two plugins for reasoning, Protégé Axiom Language (PAL) and Algernon. All the relevant information can be found at the URL http://protege.stanford.edu



Class LearningObject.Resource.Tool.Logical.CoolModes

Concrete Class Extends
Framework

Direct Instances:
None

Direct Subclasses:
None

The CoolModes Framework as proposed by UDUI

Template Slots

Slot name	Documentation	Type	Allowed Values/Classes	Cardinality	Default
:LearningObject.Id	The Learning Object's unique identifier	String		1:1	
:InputTypes	list of types that can be used as input for this tool	Class	LearningObject.Resource.Asset	0:*	
:LearningObject.Name	The Learning Object's name. This is not an identifier, but a label (for displaying purposes)	String		0:1	
:OutputTypes	the types of outcome this tool can produce	Class	LearningObject.Resource.Asset	0:*	
:Package	The Learning Object's URI	Instance	URI	0:1	
:OtherMetadata	metainformation describing this LO. It is meant to hold either standard metadata or complementary metadata necessary for Coldex objects and not collected elsewhere in the ontology	Instance	OtherMetaData	1:*	

Figure 15. Metadata for a CoolModes Assset in the Coldex System

One of the key assumptions in Coldex is that metadata can be generated automatically to a certain extent. In fact, a primary interest of this project lies in studying how tools and context information can be used for annotating learning objects in a meaningful way. Presently, many of the metadata are automatically generated by the tools after some contextual or social information, though authors can also annotate Learning Objects to suit their particular needs. As an example, the CoolModes Framework is able to supply some metadata for its outcome (that we call CoolModesAssets). Figures 16 and 17 show, respectively, a part of the metadata provided by the CoolModes Framework for a particular application (*dice game* example) and a part of its corresponding DTD.

```
<MetaData>
  <ReadOnly>false</ReadOnly>
  <CreationDate>2004-03-24 16:02:41.161</CreationDate>
  <Version>1.0</Version>
  <ParentDocument docName="" version="1.0"/>
  <Palettes>
    <Palette name="EasyDiscuss"/>
    <Palette name="DrawPalette"/>
    <Palette name="Stochastic Experiments"/>
  </Palettes>
  <Project projectId="0"/>
  <Activity activityId="0"/>
</MetaData>
```

Figure 16. Example of Metadata generated by the CoolModes Framework

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT DocRootElement (MetaData, SessionData)>
<!ELEMENT MetaData (DocName, DocDescription?, DocKeywords,
ReadOnly, CreationDate, LastChanges?, Version, Language,
Palettes, Authors, Cooperators?, Editors?, Project, Activity)>
<!ELEMENT SessionData ANY>
<!ELEMENT CreationDate (#PCDATA)>
<!ELEMENT DocDescription (#PCDATA)>
<!ELEMENT DocKeywords (Keyword+)>
<!ELEMENT DocName (#PCDATA)>
<!ELEMENT LastChanges (#PCDATA)>
<!ELEMENT Keyword (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT ReadOnly (#PCDATA)>
<!ELEMENT Version (#PCDATA)>
<!ELEMENT Palettes (Palette*)>
<!ELEMENT Palette EMPTY>
<!ATTLIST Palette
      name CDATA #REQUIRED
>
<!ELEMENT Authors (Author+)>
<!ELEMENT Author EMPTY>
<!ATTLIST Author
      login CDATA #REQUIRED
>
<!ELEMENT Cooperators (Cooperator+)>
<!ELEMENT Cooperator (#PCDATA)>
<!ELEMENT Editors (Editor+)>
<!ELEMENT Editor EMPTY>
<!ATTLIST Editor
      login CDATA #REQUIRED
>
<!ELEMENT Project EMPTY>
<!ATTLIST Project
      projectId CDATA #REQUIRED
      projectName CDATA #REQUIRED
>
<!ELEMENT Activity EMPTY>
<!ATTLIST Activity
      activityId CDATA #REQUIRED
      activityName CDATA #REQUIRED>
```

Figure 17. DTD for the Metadata generated by the CoolModes Framework

4.4 Packaging Manager Component

The LO cache is a component that uses the Packaging Manager component to conserve recently encountered objects. Given that in the repository the LOs are stored in a compressed form, in order to access them they must be uncompressed. Hence, to avoid continually uncompressing LOs, a cache is maintained of the LOs that have been used recently. The LO cache has the following functions:

- Add LOs to the cache.
- Recover LOs from the cache
- Delete LOs from the cache

The LO cache holds the object together with metadata that defines how it is to be deployed and used. An example is shown in the figure 18 where the Coldex.Uned.Chem packages three xml files together with resources, which can be either images or program files.

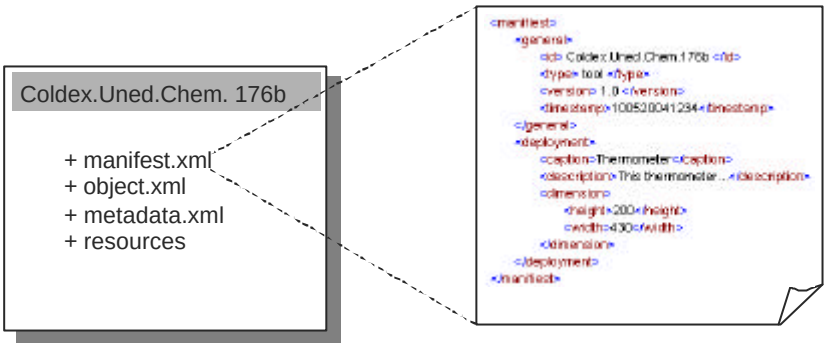


Figure 18. Example of a cached object

The relation between the Packaging Manager and the RepositoryServices can be seen in figure 19. It is made up of three separate managers, the UserManager (which enables users to be added and removed and controls access to the system), the ModelManager (which enables access to stored actors and structures), and the SocialManager (which enables access to the information about the users and the roles they will undertake).

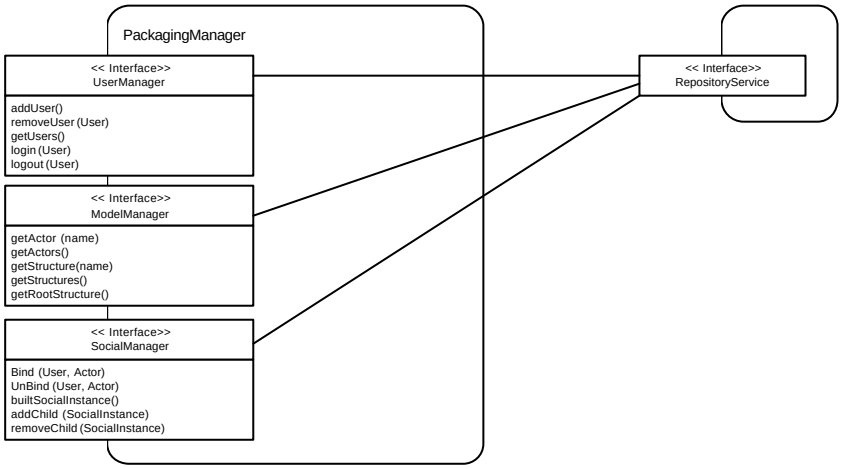


Figure 19. Relation between the Packaging Manager and the RepositoryServices

4.5 Social Manager Component

The objectives for the Social Manager Component are the following:

- Administrate the system users, register new users and delete old ones.
- Manage the working sessions of each user (login / logout)
- Manage the social model specified by the system administrators in a declarative form (XML). This will include information such as social structures that can be recognized within each model and the actors that make up one or more social structures.
- Manage the creation and deletion of the social collections in agreement with the socially specified model.

- Manage the relation between the real users and the defined actors in each social situation.

These functionalities constitute a total or partial implementation of the following use cases: 7.2.2.4, 7.2.3.1, 7.2.3.2, 7.2.3.5, 7.2.3.6, 7.2.4.9, 7.2.4.12. described in section 7.2.

This component is configured in terms of the XML that defines the structures that it manages, i.e., users, sessions and instances of social structures (communities, collectives, groups). As can be seen in figure 20, the XML specifies a model of a society, and in this example, a collection of communities. Each community is composed of a collection of groups. The society has an actor: “Manager”; each society has an actor: “teacher”; and each group has several actors: “student”

```
<?xml version="1.0" encoding="UTF-8"?>
<social>
          <actorTypes>
                    <actorType type="Manager"/>
                    <actorType type="Teacher"/>
                    <actorType type="Student"/>
          </actorTypes>

          <socialStructures root="Society">
                    <socialStructure type="Society">
                              <actors>
                                        <actor min="1" max="1" type="Manager"/>
                              </actors>
                              <subStructures>
                                        <subStructure type="Community"/>
                              </subStructures>
                    </socialStructure>
                    <socialStructure type="Community">
                              <actors>
                                        <actor min="1" max="1" type="Teacher"/>
                              </actors>
                              <subStructures>
                                        <subStructure min="1" max="3" type="Group"/>
                              </subStructures>
                    </socialStructure>
                    <socialStructure type="Group">
                              <actors>
                                        <actor min="1" max="n" type="Student"/>
                              </actors>
                    </socialStructure>
          </socialStructures>
</social>
```

Figure 20. Example of a sociey

5 COLDEX Portal

The Coldex Portal provides access to the LO repository and offers services to its users. It can be seen to be a communication and coordination medium for the virtual communities formed around it, achieved by the products that handle the LOs, using the workspaces as the underlying infrastructure. The workspaces function as the interface between the users in the virtual communities, the scenarios, and the LO in the repository. The aforementioned services refer both to the creation and maintenance of the virtual communities and the interchange of LO and

projects. In this section the virtual communities will be presented, followed by the process of creating new LO, and finally, the role of the workspaces will be considered.

5.1 Views of the Social model

What follows is an example of three screen dumps of the Coldex portal illustrating different aspects of its functionality. Firstly, in the figure 21, the social browser shows the society and its actors and the instances of its social structures.

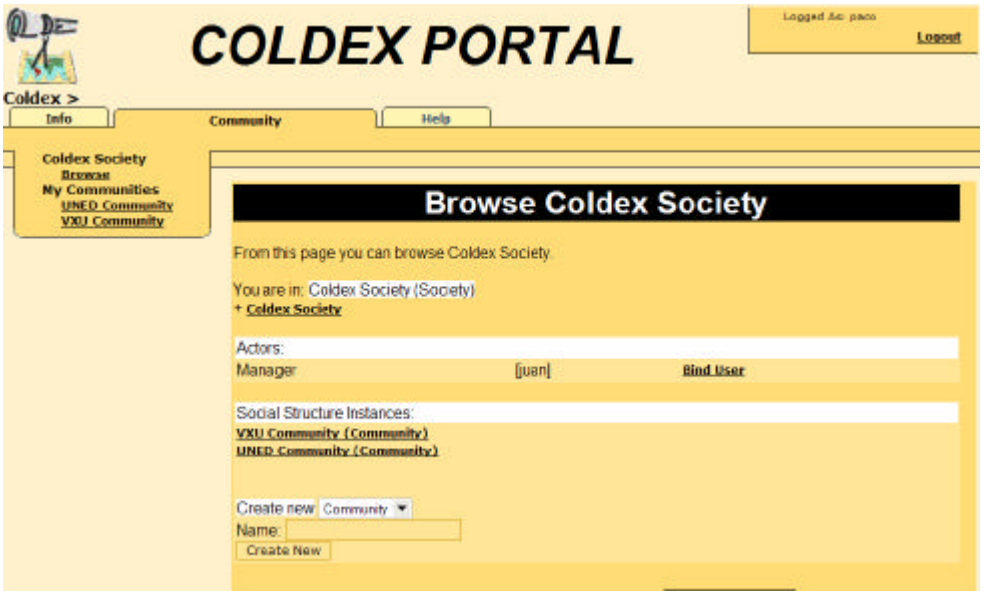


Figure 21. Snapshot presenting a society and its actors

Secondly in figure 22, a particular community can be seen, illustrating the relation between the teacher, a student, and the different social groups that have been defined.

Figure 22. View of a the “UNED” society and its actors

Thirdly and finally, in figure 23, the structure of a group can be seen, illustrating the students that form part of a particular group and its relation to the society as a whole.

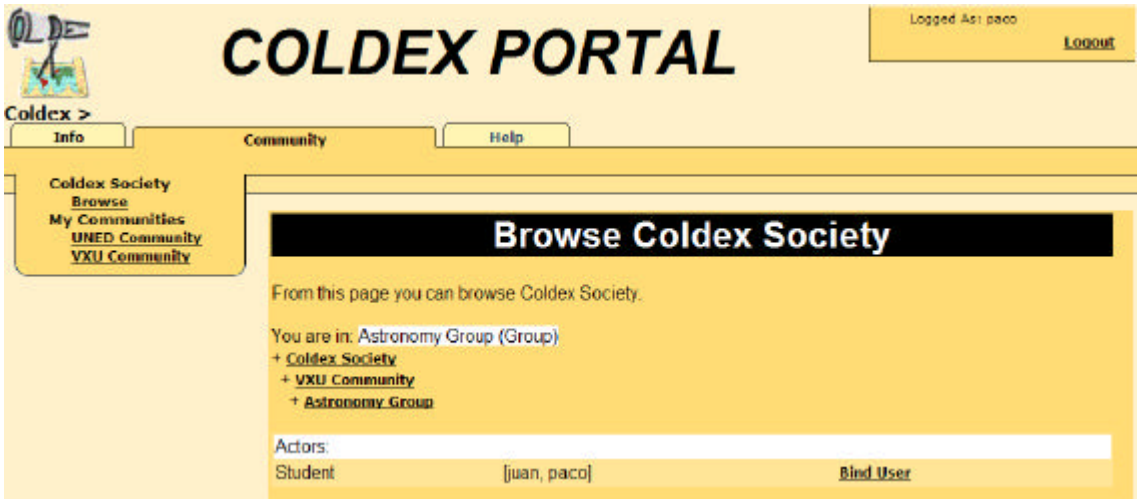


Figure 23. Structure of a gorup

5.2 Include a new type of Learning objetct

Including a **new type** of learning object into the system would require:

- Designing a form for the Learning Object **contents**. This form is aimed to ask the user with the administrator role the data that are required to deploy and use the object (for instance, the width and height for displaying the object)
- Designing a form for the (rest of the) Learning Object **metadata**, particularly, those which are specific to the new learning object (for instance, a CoolModesAsset would require the URL of the current database for CoolModes)
- Both forms can be described in a number of ways, ranging from a formal DTD to a (*non ambiguous*) description of the different fields (or attributes) which will compose the forms. It is also very important to provide explanations defining clearly the relationship between Learning Object attributes and metadata, that is, which Learning Object attributes could allow computing what metadata. Furthermore, it would be very useful having **examples of the new Learning Object types** with the right values for their attributes

As an example, lets suppose that we want to include a new type of Learning Object to reflect the *reports* the students will be writing about their learning experiences. Let be Astronomy the domain and capturing telescope images of the moon the overall project the students are working in. Then, for an activity about moon craters, each group of students involved in this project will be making a **report** as an HTML document. The new Learning Object type (**report**) could be described as follows (`attributes` are displayed in `typeface` while their *values* are written in *italics*):

- **content**
  - name: *moon craters report*
  - file reference: *Coldex/LearningObject/Reports/mooncraters.html*
  - caption: *Moon Crater Observations*
  - description: *Report on the telescope pics of the Moon Craters for the Astronomy Project made by the Group G001 at UPM*
- **other metadata**
  - author: *Group G001 at UPM*
    components:
    - *s01*
      - name: *Juan Pérez*
      - login: *jperez*
    - *s02*
      - name: *Raúl López*
      - login: *rlopez*
  - creation date: *2004.05.07 11:49:56.128* (automatically generated?)
  - format: *html* (obtained from the file reference attribute)
  - file size: *32KB* (obtained from the file reference attribute)

The **content** form would include the attributes name, file reference, caption and description while the **other metadata** form would be composed by the attributes author (which can include individual components identified by name and login), creation date (which can be generated automatically), format and size, both obtained from the content attribute file reference)

The same example, now in the form of a DTD and two XML files would be:

- DTD for the content data:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT report (name, fileref, caption?, description?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fileref (#PCDATA)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

- report.xml (i.e., an example of content for the **report** LO):

```
<report>
  <name>moon craters report</name>

fileref>Coldex/LearningObject/Reports/mooncraters.html</fileref>
  <caption>Moon Crater Observations</caption>
  <description>Report on the telescope pics of the Moon
Craters for the Astronomy Project made by the Group G001 at
UPM</description>
</report>
```

- DTD for the rest of the report metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT reportmdata (author, creation, format,
filesize)>
<!ELEMENT author (component+)>
<!ELEMENT component EMPTY>
<!ATTLIST component name NMTOKEN #IMPLIED>
<!ATTLIST component login NMTOKEN #REQUIRED>
<!ELEMENT format (#PCDATA)>
<!ELEMENT filesize (#PCDATA)>
```

- reportmdata.xml (i.e., an example of the rest of the metadata for the **report** object)

```
<reportmdata>
  <author>
    <component name="Juan PÃ©rez" login="jperez"/> ;
    <component name="RaÃºl LÃ³pez" login="rlopez"/> ;
  </author>
  <format>HTML</format>
  <filesize>32KB</filesize>
</reportmdata>
```

5.3 Adding Learning Objects Instances

As it is said before, the set of Learning Classes depends on the data made available to us. Presently, we have got a small number, which is bound to grow larger as soon as we get some feed-back. The interfaces are also subject to change in order to cope with the new objects and metadata to be added to the system

Figure 24 shows how to add an instance of an existing learning object.

5.4 Views of the workspaces

In order to access the workspace view it is necessary to select a community and also a project that belongs to it. Hence, when the 'Workspace' tab appears on the portal interface (shown in figure 25 in a circle), it is possible to have access to the relevant workspace and its options.

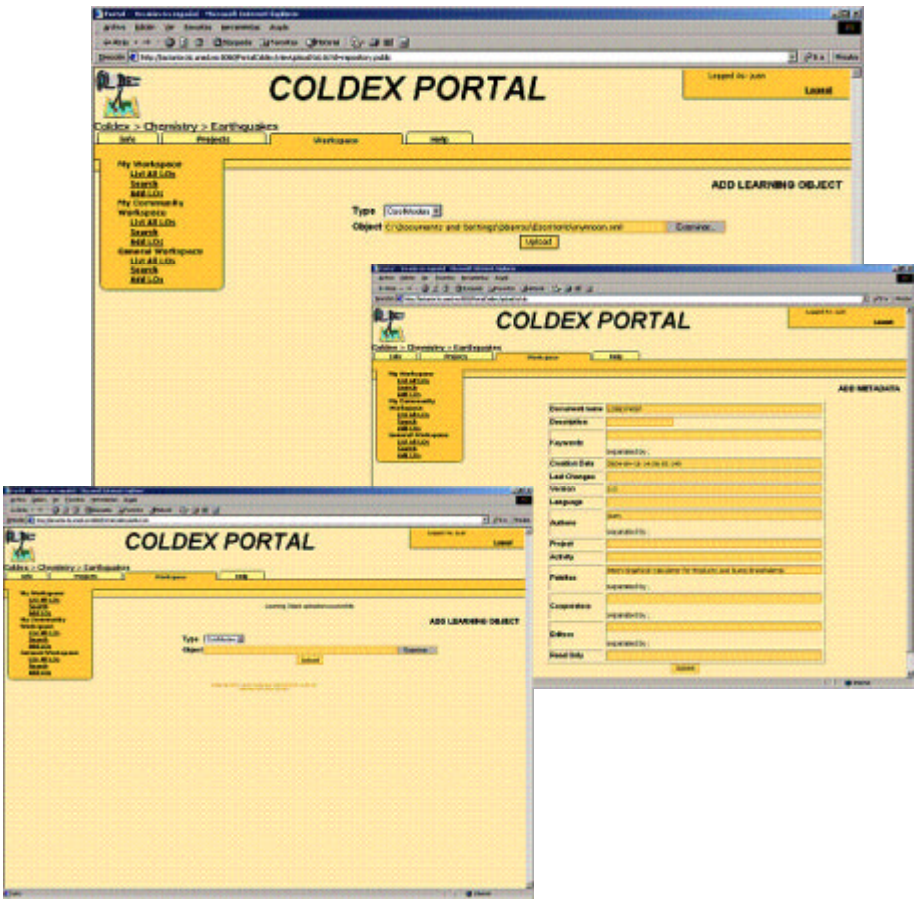


Figure 24. Adding Learning Objects

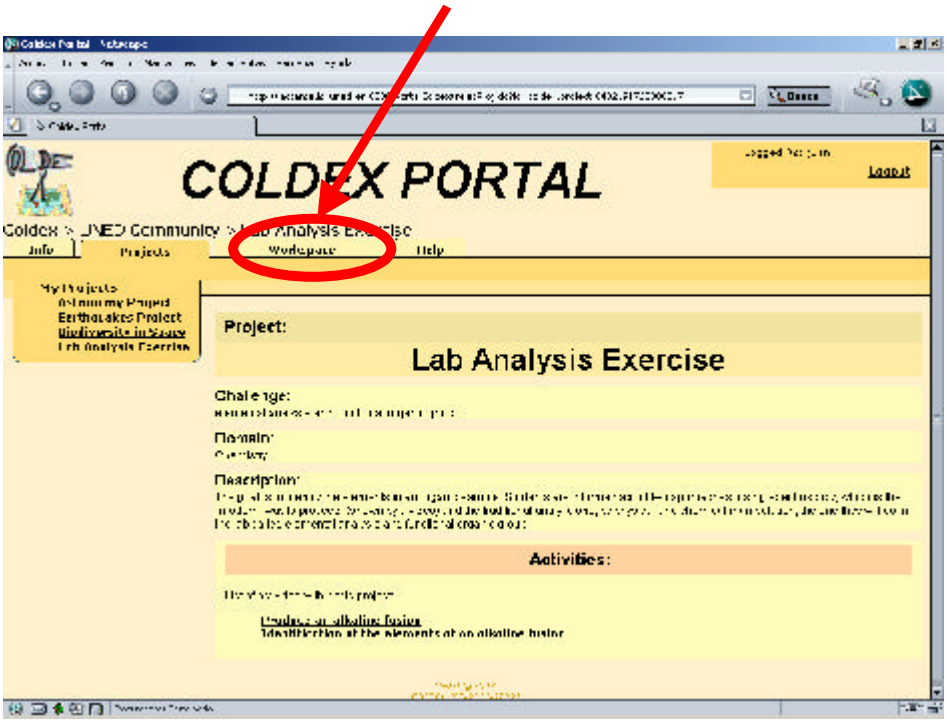


Figure 25. Access to a workspace

There are three types of workspaces: **public**, **community**, and **private**. Each type of workspace can have different LOs associated to it that have different visibility depending on the user and the community to which they belong. The three types of workspace can be defined as follows:

- The public workspace can be accessed by all users of the system together with the LOs that it contains.
- The community workspace can only be accessed by the members of the particular community
- The private workspace can only be accessed by the owner.

An example of a private workspace can be seen in figure 26, together with the LOs that belong there.

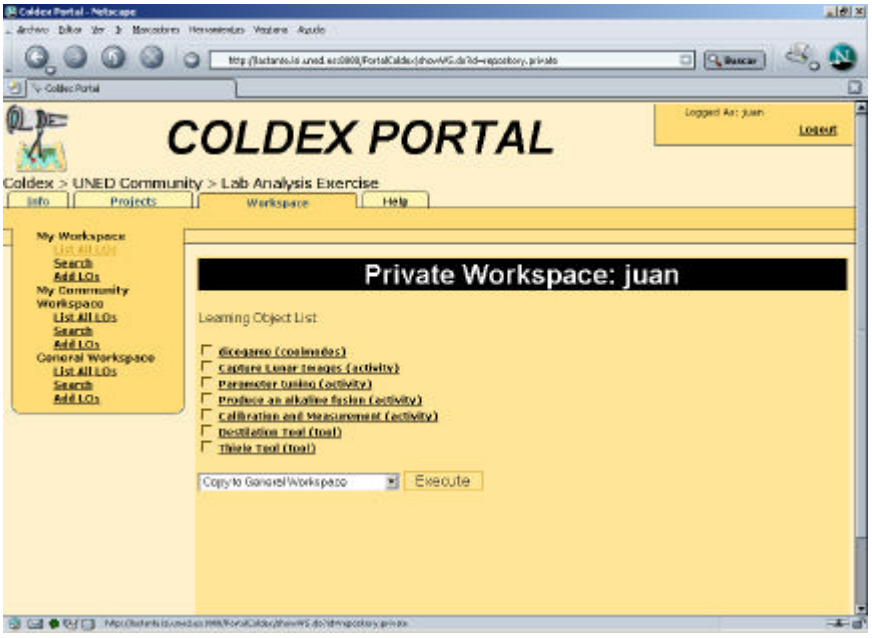


Figure 26. Example of a private workspace

There is a set of basic options available for each workspace: list the LOs that it contains, search for LOs within it (in the case that there are a very large number), and add new LOs.

In figure 25 a public workspace can be seen illustrating a list of the LOs that it contains. The options available for the LOs in the workspace are contained in a pull down menu that can be seen at the foot of the page, and can be applied to one or more LOs in a workspace. Furthermore, clicking on a particular LO will present more information about it. In figure 27 a list of some searches that can be undertaken is presented, which can search for LOs in different ways.

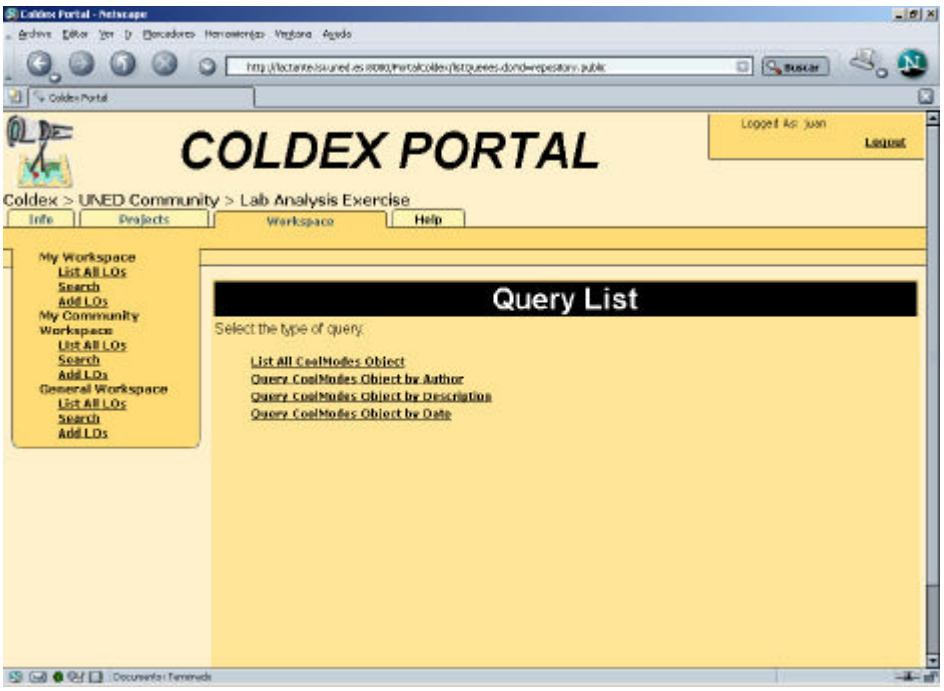


Figure 27. Some queries for searching in the workspaces

Once a particular search has been selected by clicking on it, a list of the parameters that need to be filled in appear, and once they have been filled, and the search undertaken, the system will present the results to the user, as can be seen in figure 28.

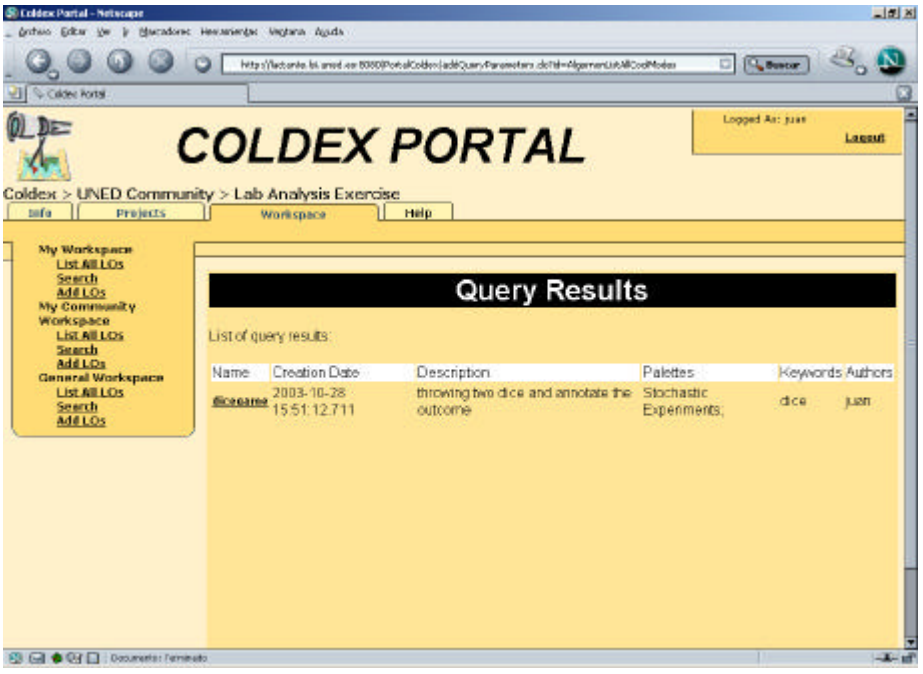


Figure 28. Results of a search

At this time the possible searches take the form of relational algebra, however, they are being extended to include more sophisticated searches that make use of the ontology. Here, the relations are reified in the ontology as first-class objects which can be dealt with just as any other object, and may include metadata. Relations can be seen as (restricted) first-order/access logic predicates or as functions. Queries will be able to check whether any given relation is held between objects, and be able to try to set conditions to satisfy given relations.

6 References

Gruber, Thomas R. (1993), "A Translation Approach to Portable Ontology Specifications", Knowledge Acquisition, Vol. 5, No. 2, Jun 1993, pp. 199-220

Booch, G.; Rumbaugh, J.; Jacobson, I. (1999) *The Unified Modeling Language User Guide*, Addison-Wesley

Brown, M. (2002) *OntoWeb Ontology-based information exchange for knowledge management and electronic commerce*. Project IST-2000-29243. Deliberable 1.2.1. http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/Deliverable1.2.1.pdf

S.Staab, J.Angele , S.Decker, M.Erdmann , A.Hotho , A.Maedche , H.-P.Schnurr, R.Studer, Y.Sure (2000) "Semantic Community Web Portals" 9th International Wold Wide Web Conference. http://www9.org/w9cdrom/134/134.html

RDF (1999) Resource Description Framework Model and Syntax Specification http://www.w3.org/TR/REC-rdf-syntax/

7 APPENDIX. Documents for analysis and design

In this appendix the documents that have been developed during the process of designing the Coldex system in its present form have been included, to help understand the architecture presented here and see how the different groups in the project have worked together.

7.1 Towards a unified vision

In this section we will try to take the first step towards a unified vision based upon the Växjo and UNED proposals (detailed in sections A1 and A2 of the appendix, respectively). That joint vision can be reached as a result of an iterative discussion and refinement process.

7.1.1 The Context

We do not see the context of the system as composed only by an environment either allowing the users to work or assisting them while doing it. We also contemplate a social and a pedagogical context where these very same users are to be taken into account.

On the technical and logistical side of the framework, there is to be an administrator responsible for the correct operation of the system, and in charge of such functions as starting or stopping the different parts that compose it (though this part could be played by a number of system administrators, each caring for a part of the whole).

On the social side of the system, there would be the learning communities, which would be composed by students and teacher(s); yet these communities would play a part in the system by themselves. We understand a learning community as a teacher and a set of learners having a common objective of educational nature A learning community would typically be split into groups of students.

On the pedagogical side of the system, there would be a number of users acting as producers of information (the teachers, but also the individual learners and the groups or communities which they compose). The teachers would organise and look after the activities to be undertaken by the learning communities within the framework.

A group known to the system (registered with it) can be involved in projects defined by a teacher. This one would create the group and define (possibly with student participation) an activity for them, which could require selecting tools for the students to work with and which would entail fetching a monitoring tool for supervising the group's work. The teacher has initially established a "list of possible" tools from which a student can subsequently choose.

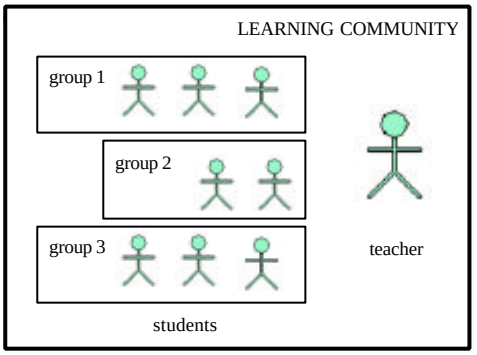


Figure 1. The Learning Community Concept

Besides this set up where the groups are structured, their components registered and the activities organised by a teacher, there could be another kind of group interacting with the system and getting profit of its features. That would be an **open community**, unstructured by nature, opportunistic with respect to its composition, for its users, though registered with the system, would not be part of a prearranged group and would not be necessarily organised or focused on a particular purpose (other than learning). This sort of community would arise spontaneously upon the unprompted interaction of the users who would constitute it.

7.1.2 The elements

The system could be roughly described as a complex educational environment where a variety of learning communities could undertake collaborative activities involving local and remote experimentation. Our aim is to unify the visions stated by Växjö and the UNED in order to obtain an initial approach to the COLDEX requirement analysis. Nevertheless, in this first version, we will try to cope with the core features of the system. So, we will not deal with a number of aspects already suggested by Växjö and which could be covered in subsequent revisions of this document, such as the role of the COLDEX partner. There will be also some issues which will be left deliberately open in order to obtain feedback on them, such as the precise definition of a DeXT or the

characterisation of the external provider of data, though we will try to suggest possible readings for all these facets as it is our purpose to work out its actual meaning.

The system external interface would be a **Portal**, which would act as an entry point and common meeting and working place for its users. The Portal could contain indices, lists and descriptions of the services it is able to provide: software tools, the development and support of teaching and activities (either organised or casual, both individual and collaborative) undertaken by learning communities, and a repository of learning materials, which would vary from mere explanations in the form of a text to complex and sophisticated software components.

Within the portal, or accessed through it, there would be a **Learning Object Repository** (LOR), a persistent storage of learning objects[4] which definition could range from single, atomic texts to complicated courseware and which purpose is to allow their reuse in as many contexts as possible. The LOR would work as a provider of content inasmuch as the material would be placed there to be got, but also and more importantly, as an organiser of this content, thanks to the mechanisms available for retrieving it. Those mechanisms, along with the objects definition, are thought to allow the use of each piece of material in many different contexts and within many different activities. Hence, from the perspective of the Portal, the LOR could be seen as the underlying structured storage mechanism used to support the former's provision of services.

Each COLDEX system will have its LOR together with a local copy of a browseable and searchable metadata **catalogue**, which reflects the contents of all the LORs. This catalogue plays a central role in the location and recuperation of objects from the different LORs. It can be seen that the LOR would be the only place where actual learning objects (that is to say, objects plus metadata) are stored within the system. There is also another intermediate repository, called the WIR (Work In progress Repository) that contains the temporary objects and data that are produced in the workspaces until they are explicitly published into the LOR. Although they are different conceptual entities, they have similar underlying implementational structures and are related in that searches can be undertaken on them both collectively, to show (for example) the temporal continuity of objects as they evolve and are published.

Hence, workspaces are a convenient metaphor for talking about what a user is actually doing in terms of what can be seen (different users will have distinct permissions, for a start, but they would also define or create, their suite of LOs). In their WS, students would undertake learning projects (designed or set up by a teacher) making use of other objects which could be retrieved from the LOR and copied to their workspace (copied in the sense of including a reference to the object). Furthermore, it would be possible to copy the contents of the WS to the desktop of the user’s computer.

---

[4] Further examples of Learning Objects may be rough data, interpreted data, project reports associated to a challenge, communities description, (remote) experiment designs, structured activities specifications (in the sense of the active document), DeXts descriptions (courseware, computer tools (software components), i.e. all sort of objects described and organised to be inspectable and reused in a wide range of teaching and learning activities

There are different workspaces to cover the diverse needs of the tasks to be carried out under the Portal. Every user (be it a teacher or an individual learner) will have a **private workspace**, which will contain that user's objects as well as any other relevant information. A user may be part of one or more groups (be they an open community or a registered, structured group, etc.). In each case, the user will have access to three other types of workspaces: firstly, the **private group workspace**, which will be accessible only to the group's members and where they can share objects, tasks, tools and results of their common work (all these aspects may be represented as learning objects in their own). In this space, it will be stored the group's 'work in progress' (as Växjö calls it). Secondly, the **shared group workspace**, which will permit the group to publish their work, make it available to the teacher. When the group's members think that their results are ready to be seen, then they would allow the teacher to access them in the group public workspace. Then, if the teacher finds the group's work suitable, it could be published (exported to the LOR). Thirdly, the **learning community workspace**, which would be an area for the learning community to share objects (data, by-products and results of the activities being undertaken by the community members).

The Workspace is not only a space to store the production, but also to integrate (for the sake of the users) the tools, as well as to organize a structure for the storage, in order to be able to retrieve things in context. So creating a workspace defines its structure, that is is given by the activities that need to be performed. In this case the procedure requires a teacher to create the group, an activity and select the tools for them.

If the design of the project only requires a workspace to be a storage place, the workspace would be defined also. In which case, the procedure is as follows:

a. The teacher creates the group, defines a structure for the workspace, and the workspace is configured as a common storage area.

b. Students select a challenge, solve problems with tools, and enter the results in the workspace.

A **project** will be assigned to a learning community by a teacher. It will be shared by all the groups within that learning community, though there could be slight variations between the groups (that is, the project features or parameters may be set up for each group). A project may have a number of phases, each of which including the performance of a variety of activities, although it is not mandatory, and hence, a project could also be defined without phases.

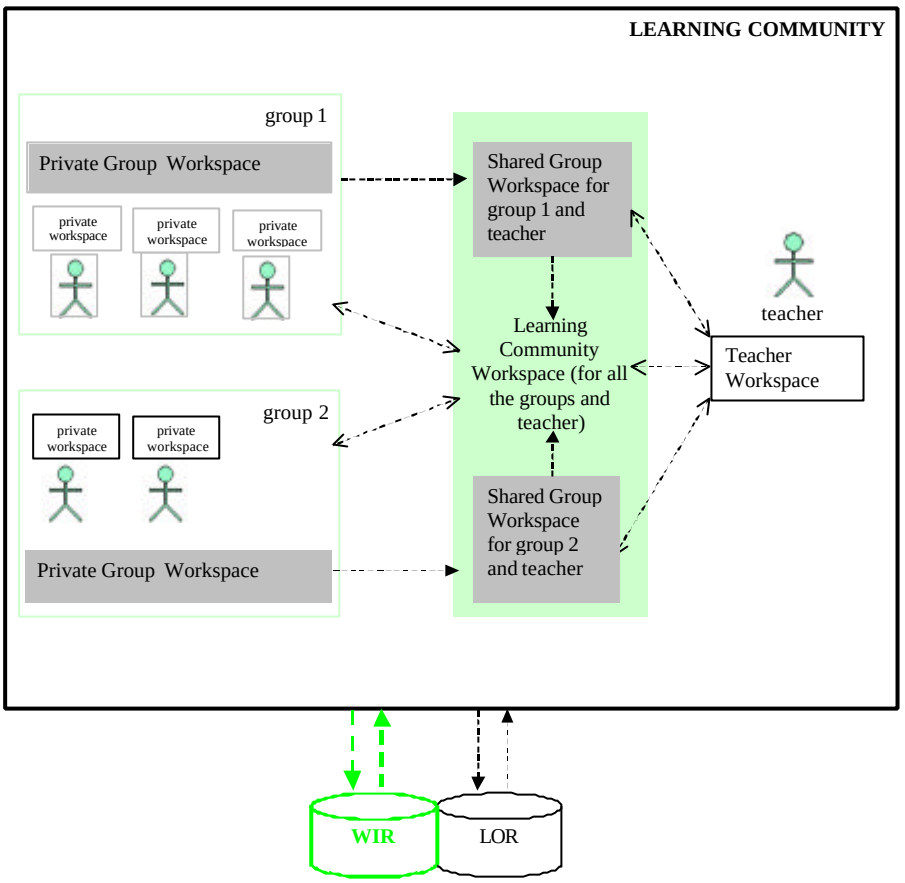


Figure 2. Workspaces hierarchy within the COLDEX system: private (learner) workspace, private group workspace, shared group workspace, teacher workspace and learning community workspace

7.2 Use Cases

In this section, we will cover those use cases which we feel that would need some clarification in order to fully define the COLDEX system. We will try to merge the use cases from the two visions that are presented in appendix 1. Whenever a use case described by Växjö is similar to one described by the UNED, they have been unified. Where this is not so, the use cases have been included in the final list. Each case is presented together with the actors involved in it, its purpose and a short description of the actions that it would include. This first joint version of both positions is intended to be a starting point for an eventual agreement.

Below, a list of actors along with their definitions is presented. Then, we will detail the use cases for each actor.

7.2.1 Actors

The main actors that take part in the system are the teacher, the individual students, the groups, the administrator, the LOR manager and the learning community. We will also describe here the role of the LOR, despite it is not an actor, because of its crucial importance for the system and, connected to it, two abstract roles: producer and consumer, who would either generate new learning objects or would use them. These

roles may be played by any user (such as a teacher, student, group or learning community), but could also be carried out by a computer program operating on its own. Though there could be more actors involved, our interest for this document lies primarily on these ones.

The user with the role "administrator" has the responsibility of looking after the system, guarantee its correct function and start or stop the system and its services as necessary. The LOR manager would operate the learning object repository, by starting, stopping, overseeing and modifying its performance as required. The teacher will take care of the formation and supervision of groups of students, who would collaborate for carrying out the assigned jobs. Both, teacher and groups make up a learning community, which adds some value to the mere aggregation students by keeping a register of the groups and their work, by supplying some communication facilities, such as a forum or a mailing list and by notifying the community members of events or of conditions that have been met.

For example, some students registered in the Chemistry course would form the Inorganic Chemistry Learning Community. Every student will have a somewhat common assignment (the lab work). Therefore, there is a community bound within the learning community. Nevertheless, each group will have to develop their own work and will be evaluated separately. Every user in the system could be a producer or a consumer depending on the job being done by this user at a particular time, who would be two new actors in the system.

The list of the actors participating in the system is given below:

- **Learning community**: a number of students with some common learning objective along with their teacher(s). The teacher would divide the community into smaller groups working in a common assignment.
- **Student:** an individual learner, part of the learning community
- **Teacher:** the responsible for organising and supervising a learning community
- **Group:** a working division of a learning community. The members of a group will collaborate for achieving the objective assigned to them by the teacher
- **Administrator:** the responsible for the system
- **LOR Manager:** the user in charge of the LOR operation and maintenance
- **Producer:** any user being able to make new LOs. This is an abstract actor including, but not necessarily limiting to, student, group, learning community or teacher
- **Consumer:** any user able to exploit a LO to work with it. It is, as the preceding one, an abstract actor.

We will not deal, in this document, with the actor "COLDEX partner" as introduced by Växjö. The work to delimit it is to be done at a later stage.

7.2.2 Administrator Use Cases

Use case: **7.2.2.1 Start System**

Actors: Administrator

Purpose: Set the system at work

Description: Turn on the system so that it can be used by all the other actors. This implies recovering the state if necessary

Use case: **7.2.2.2 Stop System**

Actors: Administrator

Purpose: Make the system cease working

Description: Turn the system off. This means making the appropriate arrangements in order for the system to continue working adequately when restarted

Use case: **7.2.2.3 Monitor System**

Actors: Administrator

Purpose: Care for the system, maintain it and tune its performance

Description: Look after the system, collect statistics about its performance and try to improve its operation

Use case: **7.2.2.4 Create LC (Learning Community)**[5]

Actors: Administrator, Learning Community

Purpose: Open a new learning community as well as its services

Description: Start the services offered by the learning community. Register the learning community with the system, that is, make a list of its components.

7.2.3 Learning Community (LC) Use Cases

Use case: **7.2.3.1 Create LC**

Actors: Administrator, Learning Community

Purpose: Open a new learning community as well as its services

Description: Start the services offered by the learning community (see Administrator's "Create LC" use case)

[5] This use case includes the “Register” of Växjö

Use case:

7.2.3.2 Configure LC

Actors: Learning Community, Teacher

Purpose: Establish the working possibilities and constraints for a learning community

Description: The teacher sets up the communication services that a learning community will offer to its components (for instance, opening a forum or set a deadline). This can be done both after the creation of the community or during its operation.

Use case:

7.2.3.3 Start Forum

Actors: Learning Community

Purpose: Begin a Forum for all the LC members and the Teacher

Description: Begin a Forum, that is, a communication space for all the LC members and the teacher

Use case:

7.2.3.4 Stop Forum

Actors: Learning Community

Purpose: Cease or discontinue an open forum

Description: Terminate or break off the operation of a forum

Use case:

7.2.3.5 Supervise LC

Actors: Teacher, Learning Community

Purpose: Oversee a LC

Description: Monitor a LC: collect data on its working, elaborate stastistics and act to amend wrong situations or to lead its members in a desired direction

Use case:

7.2.3.6 Notify situation

Actors: Learning Community

Purpose: Apprise the members of a LC of a community-related event

Description: Give a notice to every LC member as an event happens or is about to occur or when a condition is met, for instance when a deadline is approaching or has been already reached or the moment the entire LC is registered with the forum or have finished their assignments

7.2.4 Teacher Use Cases

Use case: **7.2.4.1 Create a new project**

Actors: Teacher

Purpose: Create a project for a particular LC

Description: The teacher creates a new project, that is:

Create a LC

Define the groups

Configure a LC

Define a new project

Assign a new project to the LC

Use case: **7.2.4.2 Define a new project**[6]

Actors: Teacher

Purpose: Define a framework for a project. It could suppose to use a tool or DeXT offered by COLDEX

Description: The teacher defines a new project, that is, either:

Describe, if it is needed, its phases, a challenge for the project and a DeXT. If the challenge is not selected by the teacher, it can be selected by the group at the beginning of the project

or

Define an structured project following the Active Document schema

Use case: **7.2.4.3 Define Activity**

Actors: Teacher

Purpose: The specification of an activity and its components for future use.

Description: The teacher defines an activity in terms of objectives to be achieved, the tasks to be undertaken and the resources contained in the COLDEX system that are needed. Previously defined activities can be used as templates if desired. The resulting definition is stored in the system.

[6] The definition of the term project can be found in the previous subsection "interpretations"

Use case:

7.2.4.4 Assign Project

Actors: Teacher

Purpose: Assignment of a project to a learning community or a group of users and creates an instance of the project.

Description: The teacher selects a project (by searching or navigating) from the set of existing ones in the COLDEX system and gives it to a user or group of users. Workspaces are created and allocated, as necessary

Use case:

7.2.4.5 Assign Activity

Actors: Teacher

Purpose: Assignment of an activity to a project

Description: The teacher selects an activity (previously available) in the COLDEX system and assigns it to a exisiting project

Use case:

7.2.4.6 Monitor Activity or Project

Actors: Teacher

Purpose: Control the progress of users or groups of users in a particular activity

Description: The teacher can supervise an activity or a project by checking the status of group members and the results generated, which in itself gives an indication of progress, or by using a specific monitoring tool that presents activity progress in a relevant format

Use case:

7.2.4.7 Start Project

Actors: Teacher

Purpose: Allow the learning community which have this project assigned to start working with it

Description: The teacher enables the learning community to start working with the project it has assigned

Use case:

7.2.4.8 Terminate a Project

Actors: Teacher

Purpose: Stop an ongoing project

Description: The teacher selects a project associated with a particular user or group of users and changes its status to have stopped. Associated users are subsequently unable to continue with it. If a result has been generated it can be stored in the COLDEX system.

Use case:

7.2.4.9 Define group

Actors: Teacher

Purpose: Make group and associate users

Description: The teacher can create a new group, assuming a unique group name, and select existing users from the COLDEX system and assign them to the group. Then, the teacher would add the group to a existing learning community

Use case:

7.2.4.10 Get main login

Actors: Teacher, Administrator

Purpose: To be able to use the COLDEX system as teacher

Description: The teacher asks for permission to enter the system (a login) and after a registration procedure she gets a login back.

Use case:

7.2.4.11 Communicate

Actors: Teacher

Purpose: Define the communication for the learning community

Description: The teacher activates mail, chat, discussion thread, notice board-functions to communicate with students and other colleagues in the learning community.

Use case:

7.2.4.12 Supervise ongoing group project

Actors: Teacher

Purpose: Supervise ongoing group project

Description: The teacher looks at the work that the group has stored in the "shared group workspace" and gives the group feedback through a communication channel.

Use case:

7.2.4.13 Save for publication

Actors: Teacher

Purpose: The teacher publishes the students' work.

Description: The teacher looks in the "shared group workspace" and decides that the student work is finished. If the result is suitable, it can be exported to the LC and/or LOR

Use case:

7.2.4.14 Browse Group Work

Actors: Teacher

Purpose: Navigate through the work of a group in its "shared group workspace"

Description: Have a look at a group's work (this operation could facilitate assessing this work or preparing it for publication, for instance)

Use case:

7.2.4.15 Define Resource

Actors: Teacher

Purpose: Have a new resource defined (be it a tool, a DeXT or whatever object which could help developing an activity).

Description: create and set up a new resource stores it in the LOR and makes it availabe to the COLDEX system.

Use case:

7.2.4.16 Browse Stored Resources [7]

Actors: Teacher

Purpose: The teacher looks at published resources (tools, published project, DeXTs, etc.), stored in de shared group workspace in different learning communities.

Description: Select a set of previously published projects. Diplay its contents.

Use case:

7.2.4.17 Retrieve Group Work

Teacher

[7] It unifies the use cases "Browse Resources" (UNED vision) and "Look at publications" (Växjö vision)

Purpose: Get a copy of a work published by a group

Description: Obtain a copy of the work that a group has made available to the teacher in the group's public workspace

Use case:

7.2.4.18 Evaluate Group Work

Actors: Teacher

Purpose: Assess the result or ongoing work of a group

Description: Evaluate the work that a group has done and published in the shared group worksapce

Use case:

7.2.4.19 Fetch a Resource[8]

Actors: Teacher

Purpose: Obtain a tool or a DeXT from the COLDEX system

Description: The teacher locates a tool or a DeXT in the COLDEX system by undertaking a search or by navigating through the available ones.

The available resources are been displayed in a list.

The teacher chooses one or more of them and and subsequently copies it to a workspace (that of the learning community or the shared group).

7.2.5 Group cases

Use case:

7.2.5.1 Undertake Activity

Actors: Group

Purpose: Work together to collaboratively undertake an activity

Description: The members of a group can take part in an activity following the roles assigned by the teacher. The result of the activity is identified as belonging to the group and saved in the private group workspace

Use case:

7.2.5.2 Publish Results

Actors: Group

[8] This use case considers the "Fetch resource" in the UNED vision and "Get DeXT" in the Växjö vision

Purpose: Share the results produced by a group with other users or groups

Description: Once a group has produced a result collectively, it can be published in the learning community workspace and/or in the LOR

Use case:

7.2.5.3 Retrieve Group Work

Actors: Group

Purpose: Get back work created by the group previously and stored in the COLDEX system

Description: A group can obtain a copy of work that has previously been stored in the COLDEX system, the selection of which would be based upon properties previously associated to the stored copy that could form part of a search.

Use case:

7.2.5.4 Save Group Work[9]

Actors: Group

Purpose: Store work created by the group in the COLDEX system

Description: A group can archive work they have produced in the private group workspace, defining properties such as its name, date, type, etc., which can be used in the retrieval process.

Use case:

7.2.5.5 Communicate[10]

Actors: Group

Purpose: Exchange information with other users or groups

Description: A group can communicate with other users or groups with the communication tools provide for learning community.

Use case:

7.2.5.6 Look at archived projects

Actors: Group

Purpose: The community group looks at published projects and other archived material.

[9] It corresponds to the “Save work in progress” in the Växjo vision

[10] This could include the "Global contact network" case in the Växjö diagram, though we consider it an open question

Description: The community group looks at and gets inspired by the material saved by others in the LOR

Use case: **7.2.5.7 Chose challenge**

Actors: Group

Purpose: The group choses a challenge to work with in the project

Description: The group gets a list of available challenges or makes one up by themself, and choses it by registering the challenge in their workspace

Use case: **7.2.5.8 Work with a challenge**

Actors: Group

Purpose: The group works with a previously selected challenge

Description: The group works according to the challenge they have selected

Use case: **7.2.5.9 Use Resource**

Actors: Group

Purpose: The community group uses tool or DeXTs in the COLDEX system

Description: The community group uses documents and tools for concept mapping, modelling, simulation or experimentation

Use case: **7.2.5.10 Self evaluation**

Actors: Group

Purpose: The group evaluates its work

Description: The community group follows and evaluates its own work in relation to common goals and tasks

7.2.6 Student Use Cases

Use case: **7.2.6.1 Work in private workspace**[11]

Actors: Student

[11] It considers the use case "Show personal work"

Purpose: Shows the work created by the student on a previous occasion in the private workspace and stored it in the COLDEX system, and allows him to work in the project

Description: A student can obtain a copy of work that has previously been stored in the COLDEX system (the selection of which would be based upon properties previously associated to the stored copy that could form part of a search). Then, he works in the workspace with the available tools and services.

Use case:

7.2.6.2 Save Personal Work

Actors: Student

Purpose: Store work created by the student in the COLDEX system

Description: A student can archive work they have produced to the COLDEX system, defining properties such as its name, date, type, etc., which can be used in the retrieval process. This work may be saved in the user's private workspace, in the group private workspace (by default), in the group public workspace, in the learning community workspace or in the LOR

Use case:

7.2.6.3 Communicate

Actors: Student

Purpose: Exchange information with other users

Description: A student can communicate with other users either asynchronously or synchronously. The recipient of a message could be any user (or number of them) within the learning community members.

Use case:

7.2.6.4 Undertake Activity

Actors: Student

Purpose: Work on a particular activity to achieve the objectives associated with it

Description: A student can undertake an activity by making use of the resources previously assigned to that activity (if they are already selected, though this is not mandatory) to attain its objectives. Learners may use other resources saved in the COLDEX system and available for them. Partial results may be stored in the private workspace.

Use case:

7.2.6.5 Export to Group Workspace

Actors: Student

Purpose: Share an individual result of piece of work with the members of a group within the COLDEX system

Description: A student will have work in his private workspace and will be able to select from that workspace to export a particular item into the group workspace, whereby all members of the group can access it.

Use case:

7.2.6.6 Browse group work

Actors: Student

Purpose: Inspect the results and work undertaken by other members of the group in the Group Workspace

Description: A student will be able to browse the contents of the Group Workspace for the groups in which the student belongs. The contents of the collective workspace can be classified to enable the student to be able to order work by such properties as owner, production date, type, etc.

Use case:

7.2.6.7 Register

Actors: Student

Purpose: The student asks to be registered with the system in order to gain access to it.

Description: The student requests access to the system. The teacher may grant it and then register the student with a group within a learning community.

Use case:

7.2.6.8 Retrieve archived Work

Actors: Student

Purpose: The learner looks at published projects and other archived material.

Description: The learner looks at and gets inspired by the material saved by others in the COLDEX system

Use case:

7.2.6.9 Save in group workspace

Actors: Student

Purpose: The learner transfers material from private space to (public or privave) group space.

Sescription: The learner takes material produced in her own private space and transfers it to to the group or common workspace for the other group members to see

Use case:

7.2.6.10 Self evaluation

Actors: Student

Purpose: The learner evaluates her own work

Description: The learner follows and evaluates her own work in relation to common and individual goals and tasks.

7.2.7 LOR Manager Use Cases

The use cases that the LOR Manager may perform under the role of producer or consumer are not given below but in the producer and consumer sections, right after this one.

Use case:

7.2.7.1 Start LOR

Actors: LOR Manager

Purpose: Turn on the LOR

Description: Begin the LOR operation so that it can be used by all the other actors. This implies recovering the state if necessary

Use case:

7.2.7.2 Stop LOR

Actors: LOR Manager

Purpose: Interrupt the LOR operation

Description: Turn the LOR off. This means making the appropriate arrangements in order for the system to continue working adequately when restarted

Use case:

7.2.7.3 Monitor LOR

Actors: LOR Manager

Purpose: Supervise the LOR and perform its maintenance

Description: Look after the system, collect statistics about its performance and try to improve its operation

7.2.8 Producer and Consumer Use Cases

For the following description, whenever the Actor in the use case is consumer, it is understood as including the producer (see discussion in section A2, under the Consumer subsection)

Use case: **7.2.8.1 Browse LOR Catalogue**

Actors: Consumer

Purpose: Navigate the LOR Catalogue

Description: Move through the LOR catalogue and explore it

Use case: **7.2.8.2 Search LOR Catalogue**

Actors: Consumer

Purpose: Look for LOs in the LOR Catalogue

Description: Define searching criteria and obtain a set of LOs matching them

Use case: **7.2.8.3 Export to LOR**

Actors: Producer

Purpose: Move LOs from a workspace to the LOR

Description: Publish LOs in any workspace (the producer's WS) to the LOR. This operation may require adjusting the relative sets of metadata

Use case: **7.2.8.4 Select LOs**

Actors: Consumer

Purpose: Choose a set of LOs

Description: Designate or mark a set of LOs of interest for the Consumer (by navigation through the catalogue). The operation would copy these objects to the user's WS (so, this operation would include retrieving the LOs from the LOR) and would allow partial selection of each object apart from picking up certain objects (that is, getting a part of the entire object).This operation may require adjusting the relative sets of metadata (i.e., choose the LOs metadata adequate for the consumer's purposes)

Use case: **7.2.8.5 Store LO in LOR**

Actors: Producer

Purpose:	Save a LO into the LOR (for persistent storage)
Description:	Add in permanently a LO, which was located in the producer's WS, to the LOR. This is the basic use case that would be used for saving any LO into the LOR (either for exporting from any producer's workspace or by retrieving the LO from other (external) source

Use case: **7.2.8.6 Retrieve (LO) from LOR**

Actors:	Consumer
Purpose:	Get a LO from the LOR
Description:	Get a copy of a LO from the LOR to put a workspace or provide to a software component. This is the basic use case that would be utilized whenever obtaining a LO from the LOR, be it by searching or navigation through the catalgue

Use case: **7.2.8.7 Adjust Metadata**

Actors:	Consumer, Producer
Purpose:	Convert a set of metadata into another set of metadata
Description:	Adapt the metadata of the Learning Objects to be copied so that they are consistent with those of the place where they are to be put.

7.3 The Metadata Synchronisation Service and other services

7.3.1 Technological options for the MSS

The technological options for the MSS could be implemented using one of three types: distributed Java objects, Web Services/SOAP, and messaging. Each one offers advantages for certain types of applications and disadvantages for others.

7.3.1.1 Distributed Java Objects

There are several technologies that enable objects to be accessed in a distributed way, such as RMI/IIOP, JavaSpaces, etc. As such, it would be possible to either treat the entire metadata catalogue as a distributed Java object or maintain local copies of the catalogue and use distributed objects just to transfer the changes.

While the technology has many advantages for this type of application it does not seem to be feasible to use it in a scalable, robust and guaranteed way to maintain local copies of a metadata catalogue, taking into account the topological difficulties that the network imposes (with COLDEX Server installations planned for several different countries). It is a heavy weight solution for a lightweight problem, where the COLDEX Server network will function as a loosely coupled system; something that does not require distributed object technology. If the COLDEX servers were to be used on an Intranet or on a network where performance could be guaranteed, then distributed object technology would be a candidate, in this case, where the servers are connected across the Internet, it does not seem to be.

7.3.1.2 Web Services and SOAP

Web Services, in the general meaning of the term, are services offered via the Web. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. An often-cited example of a Web service is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call.

Packaging the MSS as a Web Service does not make much sense, since each COLDEX Server installation would have to be continually polling another to see whether updates for the metadata catalogue are available. However, it has to be mentioned here to enable it to be discounted due the attention it is receiving at the moment.

Independently of the suitability of the Web Service model to package the MSS, there are further problems with Web Services that make it unsuitable here, for example:

- Web Services do not currently have a mechanism that guarantees that a service is provided. In this project the MSS has to work, always. There can never be inconsistencies in the local metadata catalogues. If a particular installation is off line, then when it comes back online, it needs to be updated transparently. If there are intermittent network problems a metadata update must always get through. IBM has proposed a reliable HTTP (HTTPR) to address requirements in this area but it has not yet been implemented.
- The data transmission mechanism of Web Services/SOAP has a series of issues related to it:
    - A SOAP envelope is an XML document that encodes the service required, the method on that service and the input parameters for the method. It then decodes the 'envelope' and invokes the real method underlying the SOAP service. As such, SOAP will consume a lot of

bandwidth as the number of service petitions grows large. Hence, scalability could be an issue for the MSS.

- SOAP processing requires memory. Building the XML strings and parsing them will use more memory and possibly leave garbage lying around.
- SOAP is not a native EJB protocol. In this project Java, J2SE and J2EE have been identified as core COLDEX technologies.

Consequently, it does not seem to make sense to use Web Services/SOAP for the MSS.

7.3.1.3 Messaging

Enterprise messaging has long been an important component of loosely coupled, reliable enterprise frameworks. Enterprise messaging frameworks enable one or more applications to communicate despite a variety of obstacles, which include: the requirement that both systems be running at the same time (synchronous communication), the need for multiple applications to receive the same message (multiple transmissions), the heterogeneity of most systems, and network failure.

Message Oriented Middleware (MOM) systems act as a conduit to handle the transmission of messages in a reliable way. In MOM systems, clients are decoupled from one another, allowing them to maintain optimum quality of service without actually having to be online all the time. Once this requirement is established, maintenance and scalability are much easier to manage.

The Java Message Service (JMS) has solved the standardization problems that existed previously with MOM systems. JMS defines the rules for message delivery in Java enterprise systems, and also declares interfaces to facilitate message exchange between application components and messaging systems. JMS offloads the responsibilities of guaranteed delivery, message notification, message durability, and all of the underlying networking and routing issues to the messaging system.

JMS supports two fundamental messaging mechanisms. The first is *point-to-point messaging*, in which a message is sent by one publisher (sender) and received by one subscriber (receiver). The second is *publish-subscribe messaging*, in which a message is sent by one or more publishers and received by one or more subscribers. While these two mechanisms are the actual foundation of JMS, many view the technology in terms of its three messaging models:

- **One-to-one messaging** is a point-to-point model. A message is sent from one JMS client (publisher) to a destination on the server known as a *queue*. Another JMS client (subscriber) can access the queue and retrieve the message from the server. Multiple messages may reside on the queue, but each message is removed upon retrieval.

- **One-to-many messaging** is a publish-subscribe model. A JMS client still publishes a message to a destination on the server, but the destination is now referred to as a *topic*. The key difference here is that messages placed in a topic include a parameter that defines the message durability (how long it should remain on the server awaiting subscribers). The message will remain on the topic until all subscribers to the topic have retrieved a copy of the message or until its durability has expired, whichever comes first.

- **Many-to-many messaging**, also a publish-subscribe model, extends *one-to-many* messaging. In addition to supporting multiple subscribers, this model also supports multiple publishers on the same topic. A good example of many-to-many messaging would be an e-mail listserve: multiple publishers can post messages on a topic, and all subscribers will receive each message.

The MSS within COLDEX can be seen to be a many-to-many messaging example. The extension of JMS into Message-driven beans (MDBs) EJB components enables messaging to be managed making use of all the benefits of J2EE servers. MDBs are stateless EJB which asynchronously process JMS messages. MDBs can receive JMS messages and process them. While a message-driven bean is responsible for processing messages, its container takes care of automatically managing the component's entire environment, including transactions, security, resources, concurrency, and message acknowledgment.

One of the most important aspects of message-driven beans is that they can consume and process messages concurrently. This capability provides a significant advantage over traditional JMS clients, which must be custom-built to manage resources, transactions, and security in a multithreaded environment. The message-driven bean containers provided by EJB manage concurrency automatically, so the bean developer can focus on the business logic of processing the messages. The MDB can receive hundreds of JMS messages from various applications and process them all at the same time, because numerous instances of the MDB can execute concurrently in the container.

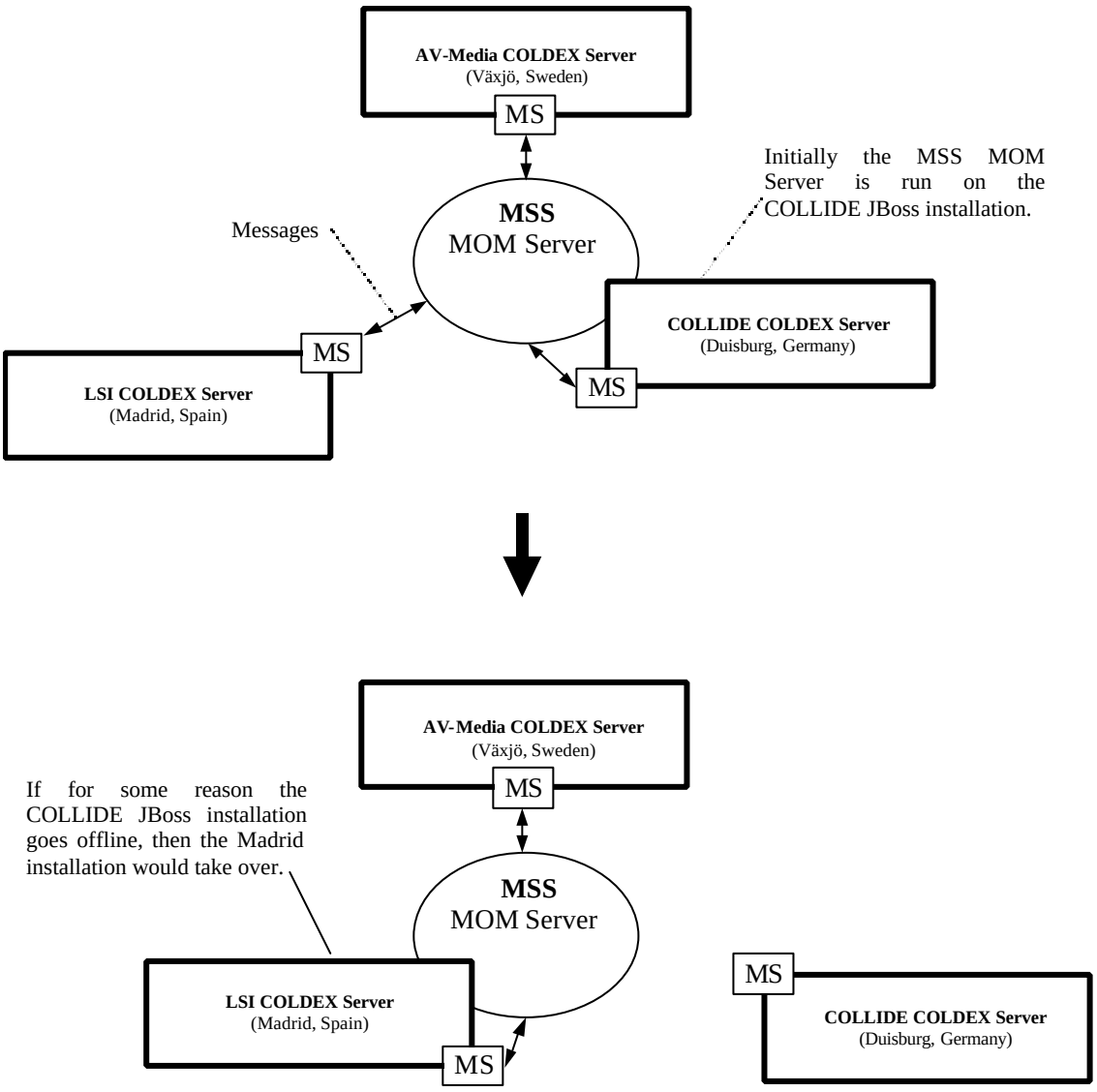
MDBs are ideal for the MSS part of the COLDEX server because they offer a robust and scalable mechanism for handling the updates of local copies of the metadata catalogue in solid way where the updates are guaranteed even though the COLDEX Server installations are loosely coupled.

7.3.2 The MSS architecture

Now that the MSS has be framed as an asynchronous messaging system built around MDB, JMS and J2EE, it is possible to present a sketch of the underlying MSS architecture that would be required to manage the messages.

In COLDEX, the proposed J2EE platform, JBoss, includes JMS MOM capabilities. In this case, any of the COLDEX Server installations could be defined a priori to act as the MOM server for the COLDEX network, and if he machine goes

down for any reason, a second server could take over the role transparently. The other COLDEX Server installations would act as clients of the MSS server. This arrangement can be seen in the following figure, where for example, initially the COLLIDE COLDEX server might hold the MSS MOM Server. Then, if the machine goes down for some reason, the UNED COLDEX server could take over the service, sending a change message to all other installations:



7.3.3 A first-draft MSS protocol

It should be noted that the way in which MDB guarantees the arrival of messages enables the MSS protocol to be greatly simplified because hand shake-based confirmation messages will not be necessary. If a message cannot get through to a client it is kept active on the MSS MOM Server until the client comes back online. Therefore, a message structure can be selected where a message name can be associated with a message body that carries the information that the recipient of the message needs to process it. Some messages would be initiated by individual COLDEX Servers and passed through to all other currently installed COLDEX Servers, and others directly from the MSS. As a draft list, the following would seem to be appropriate:

Message name	Message body	Notes
Join	COLDEX Server installation ID	When installing a new COLDEX Server a prerequisite would be knowing where the MSS MOM Server is located.
Add	New metadata	Although not a priority for the prototype, some security will be needed to stop unauthorised modification of metadata.
Update	Metadata changes	
Delete	Metadata reference	
ChangeMSS	New MSS MOM Server ID	Used by the new MSS MOM server to communicate its status to installed COLDEX servers.
InstallBase	All COLDEX Server installations	Obtain a list of all installed COLDEX servers.
NewInstall	COLDEX Server installation ID	Inform the addition of a new installed COLDEX server installation.

7.3.4 Other COLDEX services

In figure 6 of deliverable D6.1.1 a set of COLDEX protocols were defined, as well as the MSS. To summarise, these are: the **Search Service** (SS; used to undertake metadata searches), the **Learning Object Access Service** (LOAS; used to store, retrieve and modify Learning Objects stored in the LOR and their associated metadata), the **Portal & Workspace Access** (PWA, used to access to the portal and the different types of workspaces), and the **Remote Scenario Access** (RSS; used to transfer data from remote user scenarios to user workspaces (prior to incorporation in the LOR).

As was conceived in D6.1.1(draft version) these services will be synchronous client-server connections based upon HTTP or TCP. HTTP is the standard interface for the generation of Web interfaces, since a typical client will use the Web browser interface. Hence, for interactions of this type, the portal and related functions such as user communities or undertaking searches would take place via HTTP links on dynamically created Web pages using hidden page variables, re-written URLs and HTTP Sessions to pass variables and control state. Such functions as searches would be based upon data configured within an HTML form. TCP tunnels permit arbitrary objects (data, images, etc.) to be interchanged and presented to the user in a form that is coherent with the particular tool.

The Web Services/SOAP data transfer model has not been chosen, although initially attractive as a general service model, for two reasons (as well as the problems highlighted in section 1.2):

- Many client accesses to the COLDEX Server installations will take place in environments where bandwidth is a problem, for example, accesses from students working at home using modems connected to the telephone network. In this case client access to SOAP using Java requires additional SOAP libraries to be installed on the client, which are about 1.6 MB (Apache SOAP v.3.2.1 http://apache.mirrors.pair.com/ws/soap/version-2.3.1). Requiring the user to have to download and install this before being able to use the COLDEX services would be a problem.
- Following the same user scenario as the previous point, obliging a bandwidth restricted user to have to downloaded additional data (the SOAP wrappers that contain information that the user requires) will give rise to additional processing and delays that will give rise to a less reliable service.

Furthermore, Web Services are presented as a general model for offering services on the Web. However, the COLDEX services are only intended for use from the COLDEX servers, and it therefore makes little sense that any COLDEX service should be available from outside.