COLDEX

Collaborative Learning and Distributed Experimentation

Information Society Technologies Programme
Project number: IST-2001-32327

Network Specifications

Distributed Issues for COLDEX System Architecture

Deliverable Number:	D.6.1.1 Addenda 1
Contractual Date of Delivery:	
Delivery Date:	24/01/05
Version:	1.0
Lead Partner:	UNED
Contributing partners:	
Authors:	Verdejo, M.F.; Celorrio, C.; Barros, B; Velez, J.; Mayorga; J.I.
Contact:	ccelorrio@bec.uned.es

1 Introduction

1.1 Distributed COLDEX scenario

The COLDEX System is composed by several COLDEX Servers that are located in different geographical places. Figure 1 shows that each partner has their own installation of a COLDEX Server (such as portal or repository). Each repository is physically separated from the rest. The aim is to provide a distributed solution in order to supply content cohesion among different COLDEX repositories. That is, COLDEX repositories will be, partially in fact, mirrors of each other.

The distribution solution should handle the underlying data synchronisation, propagating changes from a repository to all the others. Therefore, searching will be undertaken locally, thus preventing problems due to network delays. It should be noticed that the system is distributed in the sense that learning objects are synchronised remotely between partner installations.

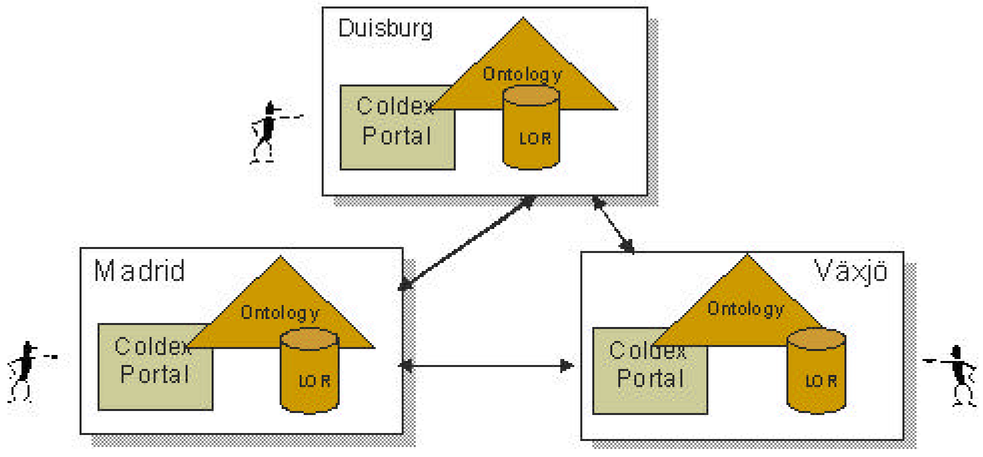

Figure 1. Architecture of the COLDEX Virtual Community

In order to design and develop an architecture for meeting the distribution requirements, it is necessary to consider the problem and all its possible solutions. This document makes a review of all the considered technological approaches to face COLDEX distribution requirements.

1.2 The Distribution Problem

The distribution scenario revolves around several factors:

- What kinds of elements must be replicated? COLDEX model includes several entities such us users, groups, actors, workspaces and learning objects. All of these objects are related with each other. A simple centralized database would be enough, should all entities had to be replicated (see section 2.1). Nevertheless, if only some elements are to be replicated (for instance, learning objects) but other element should reside locally on each server (for instance users and groups), referential integrity and data consistency should be taken into account.
- Should the distributed process be real-time or periodically launched? A real-time actualization of the system would be delivering changes all time, generating heavy network traffic. Mirroring updates ran in time intervals of several hours

could solve the problem, at the cost of not having instantly the learning objects of other partners.

2 Considered and Discarded Solutions

Different approaches have been taken into account in order to find the best solution for the distribution problem in the context of COLDEX. All of them have their own advantages and drawbacks.

2.1 Solution I. A Centralized Database

The simplest solution to grant each COLDEX partner access and visibility to the collection of LO stored in different repositories would be a centralized database schema. According to this approach, all information is stored in a central data server (sited for instance in Duisburg) and each COLDEX system stores and retrieves information from this database (see figure 1).

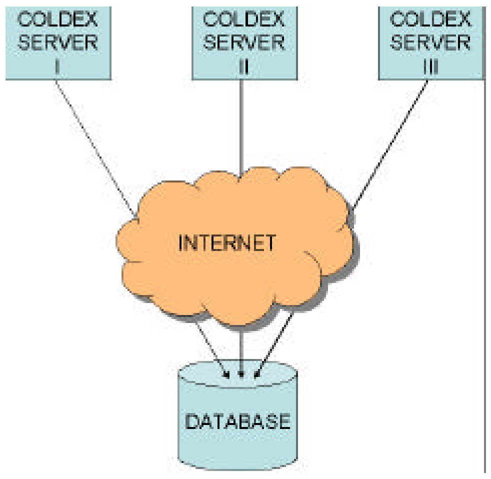

Figure 1. Centralized database solution.

2.1.1 Advantages

This solution carries out several advantages due to its simplicity:

- Simplicity. Using a single database centralized schema makes installation, configuration and management very easy and straightforward.
- Absence of integrity problems. Using a singleton database schema avoids integrity problems proper of other distributed approaches.
- Absence of Denial of Service problems. When several database servers must be updated, service denegation problems can arise on only a few of them. In this case, complex synchronization algorithms must be implemented to ensure integrity. In centralized solutions, these problems will not arise.
- High efficiency. In this approach, latency is kept to a minimum and hence the efficiency is the maximum attainable.

2.1.2 Drawbacks

The centralized solution presents several drawbacks.

- There is a unique copy of data. If database were corrupted, all data could be lost with no possible recovery solution.
- Service denial. If database server were down, then data couldn't be accessed.

2.2 Solution II. Database Server Driven Replication

MySQL 3.23.15 and higher versions feature support for one-way replication (figure 2). One server acts as the master, while one or more other servers act as slaves. The master server writes updates to its binary log files and maintains an index of the files to keep track of log rotation. These logs serve as an update record to be sent to slave servers. When a slave server connects to the master server, it informs the master of its last position within the logs since the last successfully propagated update. The slave catches up any updates that have occurred since then and then, it blocks and waits for the master notification of new updates. A slave server could also work as a master if setting up chained replication servers would be desired.

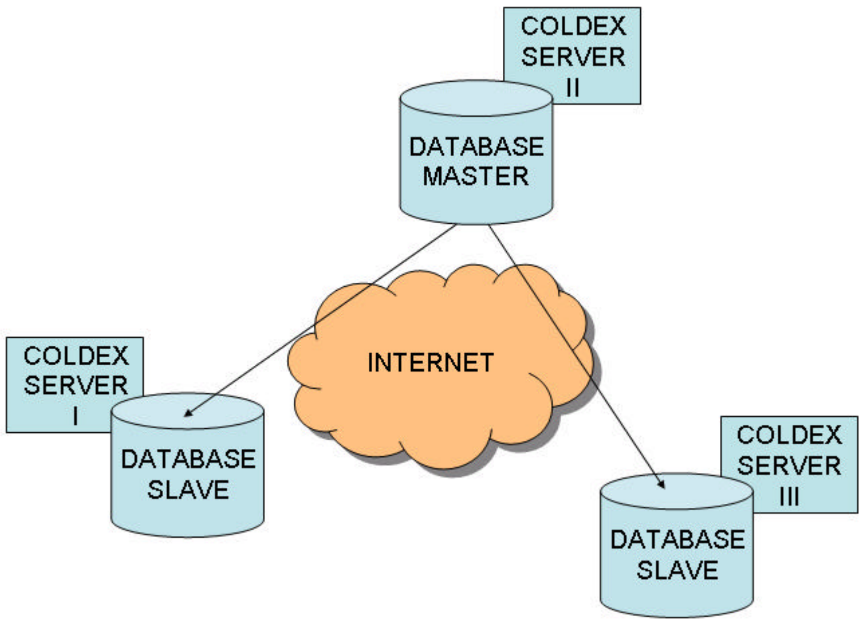

Figure 2. Database server driven solution.

Notice that while using replication, all updates to the tables that are replicated should be performed on the master server. Otherwise, care should be put to avoid conflicts between updates that users make to tables on the master and updates that they make to tables on the slave.

2.2.1 Advantages

One-way replication has benefits for robustness, speed, and system administration:

- Robustness is increased with a master/slave setup. In the event of problems with the master, you can switch to the slave as a backup
- Better response time for clients can be achieved by splitting the load for processing client queries between the master and slave servers. `SELECT` queries

may be sent to the slave to reduce the query processing load of the master. Data-modifying statements should still be sent to the master so that master and slave do not get out of sync. This load-balancing strategy is effective if non-updating queries dominate, which is the normal case

- Another benefit of using replication is that backups can be performed using a slave server without disturbing the master. The master continues processing updates while the backup is being made

2.2.2 Drawbacks

- Unidirectional updates. The main problem of this approach is that the replication schema implemented by MySQL permits defining only one master database and several slaves. Changes on the master database are propagated to slaves, but changes on slave databases are not propagated to the master.
- Master bottleneck. When master database falls down all replication system stops.

2.3 Solution III. C-JDBC Replication

C-JDBC is a persistence middleware to facilitate replication over RAID of JDBC database connections. Thus, this layer allows any Java code to transparently access a cluster of databases through JDBC. Client applications, application servers or database server software needn't be modified. It is enough ensuring that all database accesses are performed through C-JDBC.

C-JDBC provides a generic JDBC driver to be used by clients. Client drivers forward SQL requests to the C-JDBC controller that balances them on a cluster of replicated databases (reads are load-balanced and writes are broadcasted). C-JDBC can be used with any RDBMS providing a JDBC driver, that is to say, almost all existing open source and commercial databases.

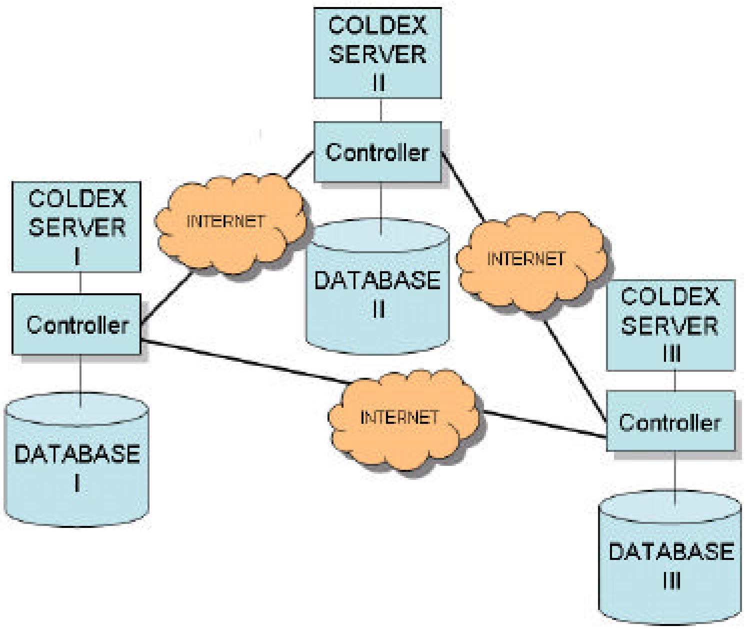

Figure 3. C-JDBC replication solution.

Though several configuration settings can be used with C-JDBC, one of them is particularly interesting to face COLDEX distribution requirements: that of having each COLDEX System to use a different database server and a C-JDBC controller. Referential integrity is guaranteed by joining all controllers in an "interaction group". Figure 3 sketches this idea.

2.3.1 Advantages

The C-JDBC bears several advantages:

- High scalability. It is possible to add new COLDEX servers on the fly without altering the normal functioning of the whole system.
- Fault tolerance. This approach tolerates database crashes and offers transparent failover using database replication techniques.
- Heterogeneous database support. As an added value this replication scheme supports for clusters of heterogeneous database engines.

2.3.2 Drawbacks

C-JDBC presents serious drawbacks:

- When C-JDBC controllers are interconnected by long distance links, they are subjects to failures which can result in cluster partitions. The problem is that data might become inconsistent between two different partitions. If both partitions evolve on the side, it is difficult to reconcile both partitions when the link comes back up.
- Management and configuration are hard tasks.

2.4 Solution IV. LDAP

LDAP (lightweight Directory Access Protocol) is an internet protocol that clients use to get information from an LDAP Server. An LDAP server consists of a database engine that stores attribute-based information. The server provides two complementary services:

- Naming services. Objects can be stored and recovered by a key
- Directory service. Objects can be bound to a collection of attributes. The directory services can recover all objects matching with a collection of those attributes (see figure 4)

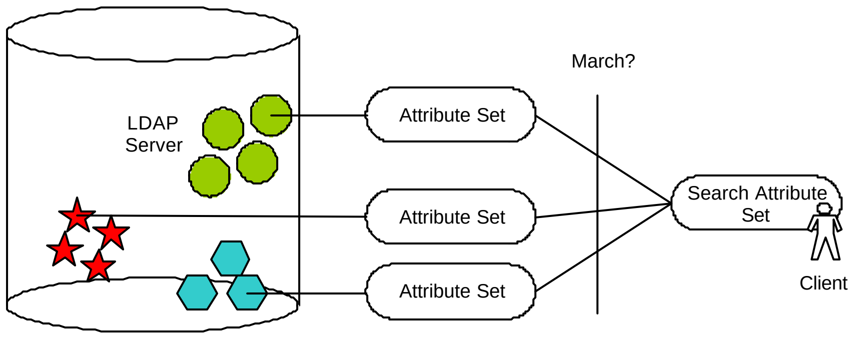

Figure 4 LDAP mechanism in the system

The OpenLDAP server provides a daemon used to propagate changes from a master LDAP server to one or more LDAP slave servers. An example master-slave configuration is shown in figure 5.

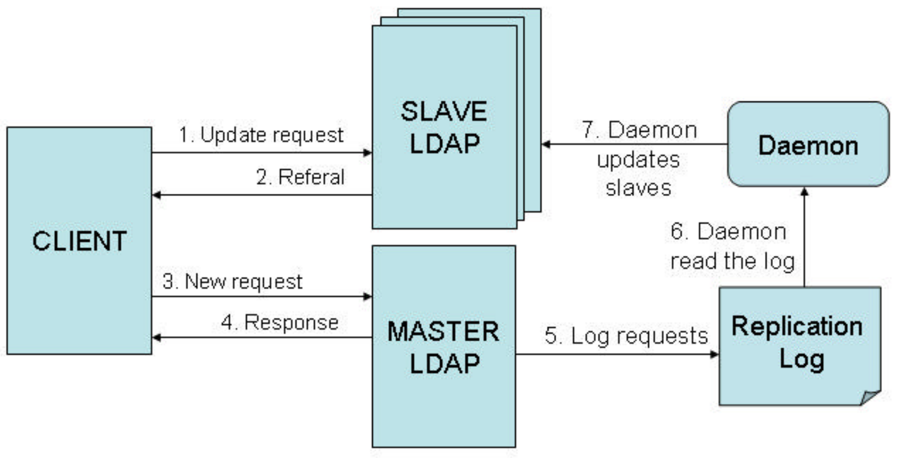

Figure 5. Open LDAP replication schema

2.4.1 Advantages

LDAP solution is particularly useful when a collection of objects is to be stored and for each one is known a unique identifier or name and a set of attributes that characterizes each repository item. This solution facilitates searching direct and attribute based searching.

- Naming services can be used to direct access and recovery of an object whose name is knows.
- Directory services permit get a collection of objects sharing a set of common properties.

2.4.2 Drawbacks

LDAP is not appropriate for COLDEX due to several reasons:

- LO are interconnected in a complex way. Thus, a simple naming/directory service is not enough to cater some search requirements.
- A relational database schema is required. For instances, COLDEX model includes workspaces, projects, activities, etc. that need to be stored and accessed in a relational way better than in simple attribute based LDAP server.

2.5 Solution V. Java Message System (JMS)

The Java Message Service (JMS) defines the standard for reliable Enterprise Messaging. Enterprise messaging, often also referred to as Messaging Oriented Middleware (MOM), is universally recognized as an essential tool for building enterprise applications. By combining Java technology with enterprise messaging, the JMS API provides a powerful tool for solving enterprise computing problems.

The Java Message Service (JMS) has solved the standardization problems that existed previously with other approaches like 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 list server: 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.

2.5.1 Advantages

JMS Technology presents two main advantages to solve COLDEX distribution problem:

- 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.

2.5.2 Drawbacks

JMS present several drawbacks:

- The general architecture to support distribution result in a very complex schema. This approach needs to many resources to be implemented and management can become very complicated.
- JMS deeply relies on EJB technology. This technology presents a very hard deployment and tuning task.

3 The Selected Solution for COLDEX

3.1 Requirements

The final COLDEX distribution approach will need to meet these requirements:

- There is a strong interest to have a replicated / redundant solution that provides access to all LOs on each one of the servers.
- There is not a critical real-time actualization need, so the replication process can be done in batch updates at the desired and programmed dates.
- Only the learning objects, along with their metadata and content, will be distributed. No other information such users, groups, projects will be replicated, and therefore it will be local to each COLDEX Server.
- Learning objects stored in users' private repositories will not be replicated, due to privacy issues.

3.2 Architectural perspective

The COLDEX Architecture is composed by several independent COLDEX Servers. Each one has a general learning object repository (LOR) which contains all learning objects (LO). These LO are created and managed by the different members and groups initialized from each Server. In addition, each LOR is decomposed logically into a collection of views or projections of the LOR (sometimes commonly named repositories). This terminology can be confused, and it is important to notice the difference between the LOR as a whole and its (logical) repositories.

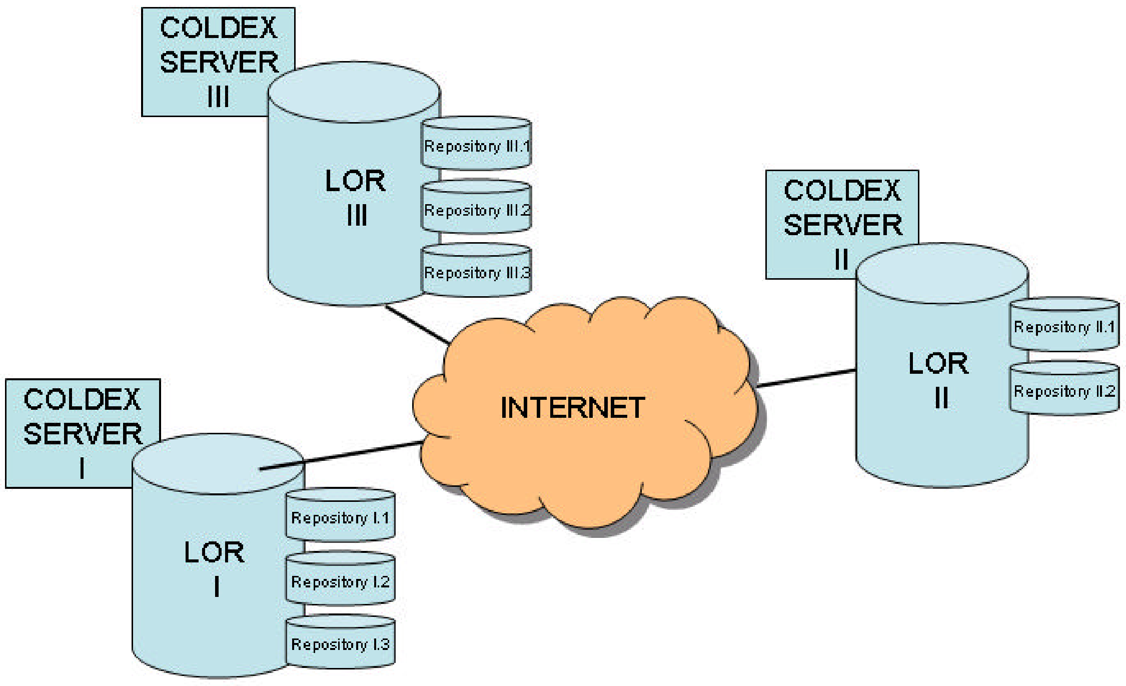

Figure 6. Architectural perspective

As can be sketched in the figure 6, 'LOR I' contains three different repositories, each one bound to different groups. The repositories store the LOs that have been created by members of the group that owns the repository.

3.2.1 Repository Content Distribution

Our approach to solve the repository content distribution consists on creating several mirroring repositories on each LOR to represent the other remote LORs within the architecture. According to this perspective the previous figure evolves to those shown in figure 7.

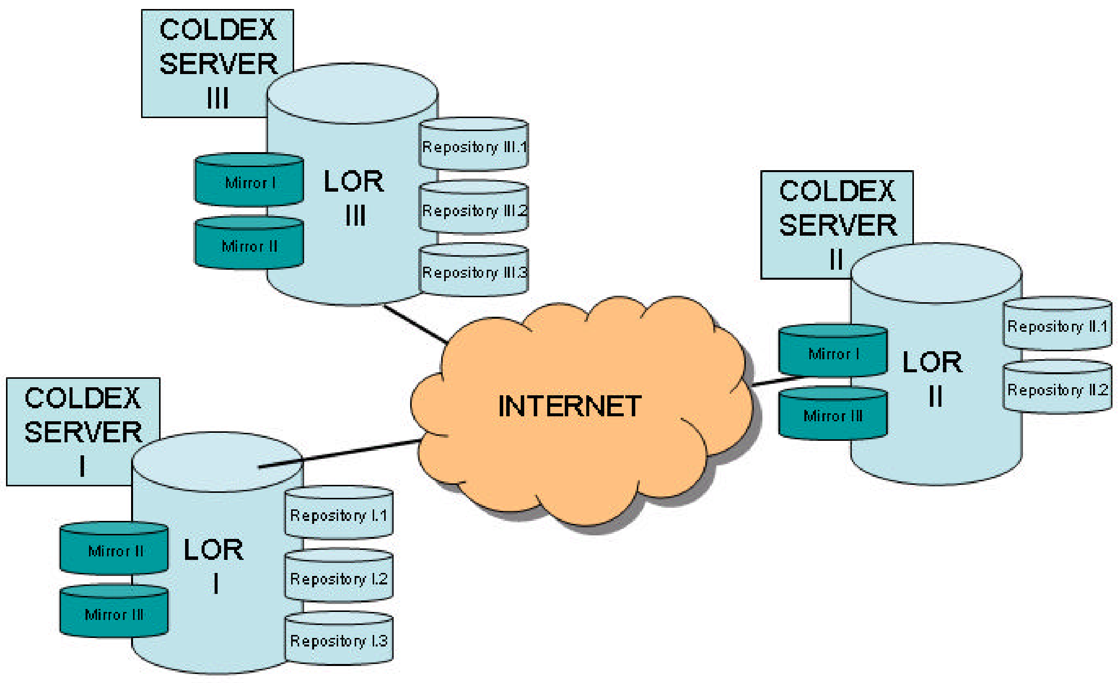

Figure 7. COLDEX Architecture with mirroring repositories

As can be seen in the figure, now the 'LOR I' contains two new special repositories, referencing to the 'LOR II' and 'LOR III'.

Every LOR has a daemon distribution to carry out the replication. This daemon periodically checks the updates on the other LORs. The checking process consists on asking the remote LORs for the list of LOs stored along with their last modification date. Then, the daemon compares this list with the LOs stored in the mirroring repository. Four different possibilities can arise:

Possibility	Cause	Required action
The LO is present in the mirroring repository and in the remote LOR, both with the same modification time	The LO have not been changed	Do nothing
The LO is present in the mirroring repository and in the remote LOR, but with different modification time	The LO metadata have been changed	Get the metadata of the remote object
The LO is present in the remote LOR but it is not present in the mirroring repository	A new LO have been added in the remote LOR	Get the whole object and add it to the mirroring repository
The LO is present in the mirroring repository, it is not present in the remote LOR	The LO have been deleted from the remote LOR	Deleted it from the mirroring repository

All the communication process between the distribution daemon and the remote LORs is done using web services. The daemon will ask for the LOs list and the remote LOR will response with a XML message (as in table X). It's important to notice that the LOs stored in the user's private repositories will not be distributed, so they should not appear on that list.

```
<entities>
      <entity id="UNED.231" modificationTime="2005-01-30 13:46"/>
      <entity id="UNED.8662" modificationTime="2005-01-29 16:12"/>
      …
</entities>
```

After retrieving the remote LO list, the daemon is responsible of asking only for the necessary data. This means that LO data traffic will be minimal, just in order to propagate the changes.

Mirroring repositories are a technical solution for the content distribution problem. The users can make use of mirroring repositories by searching, viewing or downloading LOs. Nevertheless, if they want to work with a LO stored in one of those repositories, they should copy it (or create a new version of it) to a repository of their own.

Finally, issues that must be taken into account are LO identifiers. Every LO is referenced by a unique identifier. This identifier is a read-only metadata attribute which is shown for management purposes. For instance, in order to build a LO tree version, the 'Parent' metadata field of a child LO must contain the identifier of its parent LO.

On a distributed scenario, several COLDEX Servers work in an independent way and therefore they have its own identifier generator. Hence, collision between identifiers from different COLDEX Servers may appear when replicating LOs through the COLDEX architecture. To solve this problem, the system will generate a prefix name 'Id' field. This prefix references a COLDEX Server.

For instance, valid identifiers would be 'INESC.132', 'VXU.4368', 'UDUI.324', 'UNED.9867'. (More details can be found in D.7.2.2. Metadata Definitions)

4 References

- JMS. http://java.sun.com/products/jms/
- OpenLDAP. http://www.openldap.org/
- MySQL. http://www.mysql.com/
- C-JDBC. http://c-jdbc.objectweb.org/
- Deliverable web services
- AXIS Web Services. http://ws.apache.org/axis/