The COLDEX Metadata Synchronisation Service (MSS) and other services

LTCS Group, UNED, October 2003

In section 8 of deliverable D6.1.1 it was proposed that all metadata searches should be undertaken on a local copy of the metadata catalogue stored on each COLDEX Server installation (where a user is actually connected and working), and not via a search of a single metadata catalogue distributed across all the COLDEX servers. This will greatly improve access speed and reliability by limiting the vagaries of network access and delays[1]. However, as was noted in D6.1.1, additional software will be needed to synchronise the metadata catalogues and ensure that at any moment in time, all installed COLDEX servers have the up to date catalogue. This obviously has to be done in a robust, secure and scalable manner, as per the indications of Task 6.2 in WP6. To this end, a Metadata Synchronisation Service (henceforth, MSS) was defined. This document has four objectives: firstly, to present the technological options for implementing the MSS, secondly, present a schematic architecture for the MSS based upon the chosen technical infrastructure, and thirdly, define a first-draft MSS protocol. Fourthly and finally, the other COLDEX services presented in D6.1.1 are briefly discussed.

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.

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.

[1]Anyone who has used P2P file sharing software will be aware of some of the problems of undertaking distributed searches.

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.

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.

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.

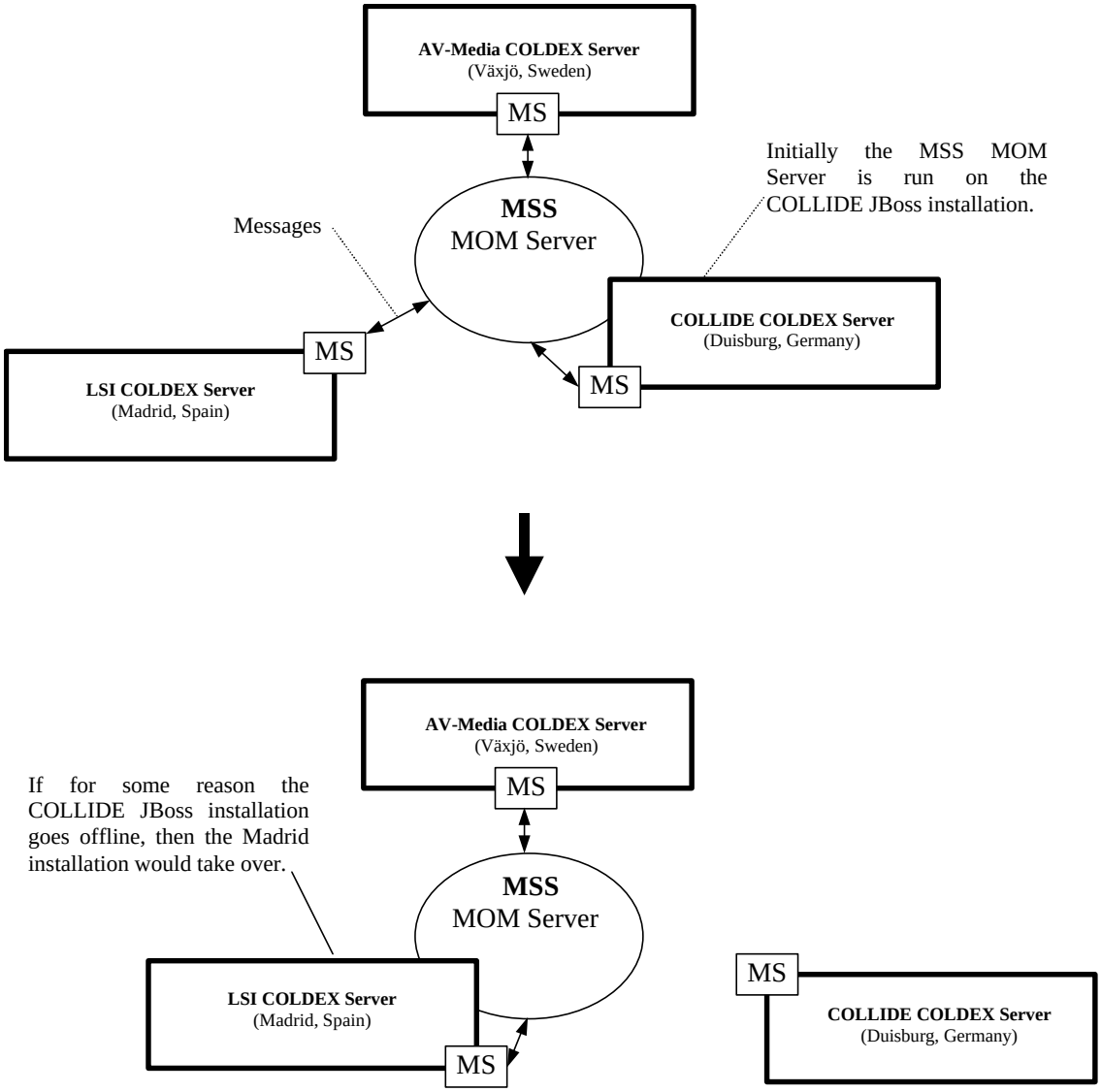
One of the most important aspects of message-driven beans is that they can consume and process messages concurrently. This capability provides a significant advantage over traditional JMS clients, which must be custom-built to manage resources, transactions, and security in a multithreaded environment. 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.

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 the 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:



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.

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 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.