USING XML TECHNOLOGIES TO FIT TO THE OAIS MODEL : A CASE STUDY

* Thierry Levoir
Tél : 05.61.27.42.50 Fax : 05.61.27.30.84
Thierry.Levoir@cnes.fr

* Marco Freschi,
Tél : 05.61.27.40.83 Fax : 05.61.27.30.84
Marco.Freschi@cnes.fr

*** CNES Centre National d'Etudes Spatiales**
18 Avenue Edouard Belin,
31401 Toulouse Cedex 4

Abstract

The OAIS (Open Archival Information System) Reference Model offers a real help to design an archive. It includes many services such as data ingest, data management, administration, access systems etc…

In such system we need features like platform independence, human readable format of data, extensibility, maintainability, … XML and all the related "new" technologies seem to provide a way to do that.

However, it is sometimes very difficult to understand where everything really fits. Many different architectures may be defined.

The article takes cue on the needed to update an existing data center. It presents the role of each single object inside of a data center and what are (or are not) the benefits using XML technologies.

Introduction

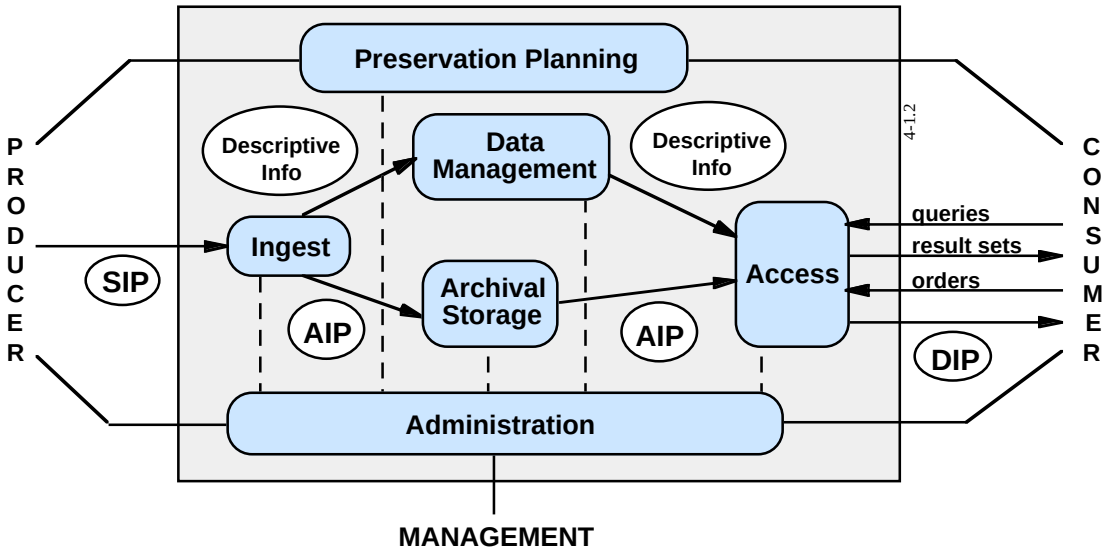
The OAIS (Open Archival Information System) Reference Model provides a framework to create an archive (consisting of an organization of people and systems, that has accepted the responsibility to preserve information and make it available for a Designated Community). It offers also a real help to design: ingest, data management, administration and data access systems.

XML stands for eXtensible Markup Language. XML starts as a way to mark up content, but it soon became clear that XML also provided a way to describe structured and semi-structured data thus making the usage as a data storage and interchange format. Many related languages, formats, technologies like SOAP, XML Query, XML-RPC, WSDL, Schema, … are still coming to provide solutions to almost all problems!

With such technologies, we can define many different architectures. Due to the vastness of the problem, it is quite difficult to describe all the possible solutions, so we intend to describe a possible architecture of a system where, the organization of data and their usage, is defined in accordance with the OAIS reference model.

This study takes cue on the needed to update an existing data center, providing some features like platform independence, human readable format of data and easy extensibility for new type of data. All these advantages seem to be supplied by XML and Java. XML and Java together can certainly be used to create some very interesting applications from application servers to better searchable web sites. However, it is sometimes very difficult to understand where everything really fits. We attempt to clarify the role of each single object inside of a data center, providing as result, the complete description of a system including its architecture.

OAIS Functional Entities



Ingest: Services and functions to accept Submission Information Packages (SIPs) from Producers and prepare the contents for storage and management within the archive

Archival Storage: Services and functions for the storage, maintenance and retrieval of Archival Information Packages

Data Management: Services and functions for populating, maintaining, and accessing both descriptive information which identifies and documents archive holdings and internal archive administrative data.

Preservation Planning : This entity monitors the environment of the OAIS and provides recommendations to ensure that the information stored in the OAIS remain accessible to the Designated User Community over the long term

Administration: Manages the overall operation of the archive system

Access: Supports consumers in determining the existence, description, location and availability of information stored in the OAIS and allowing consumers to request and receive information products

Implementation Requirements

For each component described in the previous picture we study a possible solution using new technologies. The complexity of the system shows us a wide range of possible

approaches according to the different point of views. However it is clear that the following requirements has to be met:

- **Programming Productivity:** a direct adoption of new technologies is insufficient unless they are properly used to their potential and appropriately integrated with other relevant technologies. So the choice of the tools must be analyzed also in the application general context.

- **Reliability and Availability:** the system is often used for archiving and retrieval critical data and a good reliability in every condition and a good handling of the transactions can guarantee a success of the system itself.

- **Security:** one of the major goal in the data preservation is to make secure the data exchange between user and archiving system, in particular the use of internet and the web technologies make this goal a prime concern.

- **Scalability:** in the development of system is always considered a factor of growth to meet new demands and new concepts. For this reason the system has to consider future extension both in its operation and in its user utilization.

- **Integration:** the development has to consider, when it exists, existing data even if it is old and outdated. Often it is not an easy target but, the ability to combine old and new technologies, can help to save time and resources in the migration phases.

On the base of these requirements we intend to develop a system that offers both producer and consumer the opportunity to work through internet. The idea is to create services, that will be integrated together to obtain an efficient system. The concept of creating services that can be accessed over the Web gave rise to a new term called web service. In other words our system offers the user some components accessible via standard web protocol.

Remote Customers and SOAP

Our system presents two interfaces with the external world. The former is used by the producers and it represents the way to insert new data in the system (ingest); the latter, instead, is used by consumers that extract data from the system for their purposes (access). Both producers and consumers can be remote clients, so it is necessary to find a suitable protocol to exchange structured and typed information on the web and at the same time lightweight and simple. Moreover, the protocol is based on XML and this fact gives us the first input on the power of this new technology. SOAP is a way to describe a messaging format for machine-to-machine communication. It consists of three main parts:

- **SOAP Envelope** that collect all the info about the recipient and the message itself.

- **SOAP Header** that defines header information

- **SOAP body** that contains call and response information

We decided to use a standard HTTP internet protocol even if SOAP binds other formats. The major goal in the design of SOAP is to allow an easy creation of interoperable distributed web services, providing easy access to objects. One of the most evident advantages in the service description with SOAP is due to the use of XML; in fact the

experience shows, it is much easier to describe services in XML rather than CORBA or RMI. For example, a simple request of data associated with the mission *Interball* and experiment *Hyperboloid* can be described by the following simplified SOAP XML document:

```
<soap:Envelope>
  <soap:Body>
    <GetData>
      <Operation>Retrieval</Operation>
       <Mission>Interball</Mission>
       <Experiment>Hyperboloid</Experiment>
    </GetData>
  </soap:Body>
</soap:Envelope>
```

The <Envelope> element is the root element of a SOAP message and it defines the XML document as a SOAP message. The <Body> element is mandatory and it contains the real SOAP message: in our case <GetData>, <Operation> <Mission> and <Experiment> elements are “application specific” and they are not a part of SOAP standard. The <Body> element may contain a <Fault> element. The <Fault> element is used to provide information about errors that occurs while processing the message. By nature this element can only appear in answers (response messages). Anyway SOAP was not the only candidate for our approach but a study comparison with the other solutions shown an effective list of advantages. A prime candidate for the content of the requests was XML. However, generating and parsing the XML, both on the client and on the server, can be tedious and error prone. An alternative was to use remote method invocation (RMI). But RMI has its own drawbacks, it forces all callers to be Java compliant, and it compromises scalability by requiring a "live" remote object. We were looking for something easy to use, flexible and scalable and most of the keys addressed SOAP as solution. Another key advantage of SOAP is that it does not require a permanent connection between computers. SOAP can ride with other HTTP traffic, such as Web pages, and SOAP does not require the network administrator to perform any additional setup or maintenance. To resume the analysis we can conclude that SOAP is a simple protocol that supports the exchange of objects but doesn't support remote invocation of objects. The big advantage of SOAP is its openness: Because it is built on XML and designed to use standard transports like HTTP and SMTP, any OS that supports those standards is available as a platform for SOAP development and deployment.

Data Management and XML Database

The Data Management represents another critical block of the OAIS model. A good choice of support and data representation makes a system more efficient both in term of performance and cost. Nowadays the market offers a great number of products especially in the area of relational and object models, so an easy choice should have been to adapt our data to an existing model. One thing was clear the data traveled in XML format both in input and output and a conversion had to be done to map the XML documents to whatever other model. We found the bibliography treating this problem not really well made and often specialized for particular applications. Again XML offers alternatives but it was important to know if that solution could be considered applicable. The starting point was to study the typology of exchanging data, and try to classify our working data, basically the exchanging data can be grouped in two categories:

- **Data-Centric Documents** they use XML as a data transport. They are designed for machine consumption. It is not important for the application or the DB that the data is stored in an XML documents.

- **Document-Centric Documents** they are documents that are designed for human consumption. They are characterized by less regular or irregular structure and large grained data.

In our case the data are often documents, images and normally this kind of data are not highly structured, so also in this case XML seems to be useful. The native XML databases offer several advantages. First of all when the data is semi-structured (it means it has a regular structure, but the structure is variable) a use of relational model implies either a large number of columns with null values or a large number of tables, which make the system inefficient and costful in term of space. A second advantage in the use of XML database resides on the retrieval speed. Using XML database and an accurate policy of storing, the XML database shows to be faster than relational and object databases. The reason for this is that some strategies used by native XML databases store entire documents together physically or use physical (rather than logical) pointers between the parts of the document. This fact allows the user to perform the retrieval without slower logic joins. This fact has, however, an obvious drawback: the increased speed applies only when retrieving data in the order it is stored on disk. This can cause problems if the user needs a different view of data the performances can decrease and to be worse than in a relational or object database. In our system the views are well defined so the real queries produced both the producer and the consumer are predictable. Before continuing in the possible advantages it will better to see the generic features offered by a native XML Database and how to work with.

A native XML database...

- Defines a (logical) model for an XML document (as opposed to the data in that document) and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order.

- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

An important element to know when we work with native XML database is the way to storage data

XML Storage

Native XML Databases store XML documents as a unit and will create a model that is closely aligned with XML or one of XML's related technologies like DOM. This model includes arbitrary levels of nesting and complexity, as well as complete support for mixed content and semi-structured data. This model is automatically mapped by the Native XML Database into the underlying storage mechanism. The mapping used will insure that the XML specific model of the data is maintained. Once the data is stored we has to continue to use the XML Database tools if you expect to see a useful representation of the data. For instance, deciding to use an XML Database that sits on

top of a relational database, accessing the data tables directly using SQL would not be as useful as you might expect. The reason for this is simply that the data you will see is the model of the XML document (i.e. elements and attributes) rather then the business entities that the data represents. The business entity model exists within the XML document's domain, not within the domain of the underlying data storage system. To work with the data, you work with it as XML.

Collections

An Important concept provided by XML Databases is the *collection*. The collection represents a group of XML documents that for this nature can be assimilable. Xml Databases manage collections of documents, allowing you to query and manipulate those documents as a set. This is very similar to the relational concept of a table. There is a slightly but important difference between the collection and the table concept, that is not all native XML databases require a schema to be associated with a collection. This means that you can store any XML document in the collection, regardless of schema.

Queries

XPath is the current native XML Database query language of choice. In order to function as a database query language, XPath is extended slightly to allow queries across collections of documents. Unfortunately, XPath wasn't really designed as a database query language and comes up short in several ways when it's used as one.

Some of the more glaring XPath limitations include a lack of grouping, sorting, cross document joins, and support for data types. Because of these issues XPath needs to be expanded as part of a more comprehensive language. Many of the issues can be resolved by utilizing XSLT to fill in the holes, but a more database-oriented language is under development, in the form of XQuery. Several vendors have already begun to release prototype XQuery implementations for use with their databases.

To improve the performance of queries, XML Databases support the creation of indices on the data stored in collections. These indices can be used to improve the speed of query execution dramatically. The details of what can be indexed and how the indices are created will vary widely between products, but most support the feature in some form.

Conclusion

We have focused on the ingest, access and data-management services because, it's for that we can see easily the benefits of XML. However when a project has to begin, we have to understand that the use of XML is not always the good choice. We have to avoid to use it, only to use a fashion language !