An XML Architecture for High-Performance Web-Based Analysis of Remote-Sensing Archives

G. Aloisio,
Universita degli Studi di Lecce, Lecce, Italy

G. Milillo,
Agenzia Spatiale Italiana, Frascati, Italy

R. D. Williams,
California Institute of Technology, Pasadena, California

Abstract

We introduce XML (eXtensible Markup Language) as a simple, flexible, and powerful way for computers to exchange metadata and control information, not only with humans, but also with each other. We describe an existing system that delivers customized remote-sensing data products to web-connected clients, and what more is required to support supervised, on-demand processing of the data. We discuss how such an architecture can be used for these purposes, and how it may interoperate with existing parallel SAR processing architectures, or with a GIS application server. Applications of such a system include agriculture, ecology, geophysics, glaciology and hazard-monitoring.

1. Introduction

1.1. XML and the Web

The world wide web has emerged as an extraordinarily successful medium for the transfer of information between computers and people. To exaggerate only slightly, any machine can be used to serve any kind of data with the confident assumption that it can be used by anyone in the world with a computer and an Internet connection. The complexity of the data and its production has increased rapidly, from static pages to complex, dynamically-generated sessions driven by databases. The HTML language, both for information presentation and input forms, together with GET and POST queries have become universal standards in human-computer interaction.

But when computers exchange information with other computers, the story is not so clear. For years there have been proprietary systems for exchange of objects and structured data, but the adoption of such a system would mean locking in to a particular software vendor, which is viewed as undesirable to the community of academic and government science. CORBA (Common Object Request Broker Architecture) is an open alternative, but the infrastructure investment is large, and the learning curve steep. Recently Java RMI (Remote Method Invocation) has been popular: it is easier to use than CORBA because applications may only be in one language.

In choosing how the computers of our system exchange information, there are certain trade-offs to be considered, for example, between efficiency and flexibility. The efficiency of the communication is maximized by bulk binary transfers where sender and receiver know

everything about the transfer before it begins; whereas the flexibility is maximized by using ASCII or Unicode text, with redundancy, syntax checks, and frequent flushing of streams. We believe that the key to the efficiency/flexibility trade-off is to separate command, control and diagnostic streams from the bulk binary data streams. We will use XML for the former, and for the latter conceptually distinct streams, perhaps raw sockets, HTTP, or advanced parallel streams.

Another trade-off is between capability and simplicity. Capability is maximized by allowing for multiple languages, transfer of arbitrary objects, and a large API; simplicity comes from small, human-readable transactions that are restricted in scope to a mental model that is easy to grasp. In managing a group of people collaborating on a software project, it is always important to be clear about interfaces between different people's code right from the start. A simple, transparent exchange mechanism such as XML facilitates this. We believe that XML will play a major role in enabling computers to communicate universally with other computers. While it cannot (some would say should not) exchange general binary objects, but only text-based documents, it is both simple and powerful, and its syntactic similarity to HTML ensures universal adoption. XML enables a new generation of web services that are not meant for humans to use directly, but rather to be used by other services. We hope that as our project expands in scope, we can continue to extend our markup language to deal with increased complexity, and to interoperate with other digital libraries, but at the same time retaining simplicity and clarity.

1.2. Synthetic Aperture Radar

In this paper we propose an architecture for an extensible, scalable digital library, a library that allows interactive processing, in addition to retrieval of the information in the library. Several such projects are surveyed and evaluated in [1]. As a demonstration of our technology, we will make a prototype library of remote-sensing images, specifically Synthetic Aperture Radar (SAR) images covering the Puglia region of Italy.

SAR technology is gaining attention as data continues to arrive from existing missions, and plans are made for archiving the exponentially-increasing quantities of data from future missions. A good overview may be found in [2], but briefly, an Earth satellite illuminates the surface of the Earth with radar in the wavelength range 5 - 25 cm, and the amplitude, polarization, and phase of the return signal is measured. With considerable careful computation, an image can be reconstructed. A great advantage of SAR over other remote-sensing technologies is that it can penetrate cloud, vegetation, and dry sand — 'seeing below the surface'.

We shall describe a system for interactive retrieval and analysis of SAR remote-sensing data, using XML as the communication fabric between independent services. First we describe the nature of the data and our current retrieval tools, then the new architecture, the nature of its component services, and the XML communication fabric that connects them. Finally, we present some issues related to the use of the XML-based system to provide automatically some of the functionalities of an operative center of remote sensing.

1.3. Distributed Objects

In this paper, the architecture we propose is based on the paradigm of distributed objects and distributed services, and it is organized into the now familiar three-tier view of client, middle and backend services (see section 3.6). The new feature, however, is the idea of using XML as a

language for exchanging objects between services, because it is an open, human-readable, and increasingly popular common language base for distributed applications. While the use of XML requires objects to be serialized into text streams (with binary data delivered from a web, or other type of server), this has the advantage that it can be read and edited with increasingly common tools. Another advantage, due to the variety of parsers available, is that different users can write code with different languages without the complexity of Corba.

2. Data Retrieval

2.1. The Synthetic Aperture Radar Atlas

SARA (Synthetic Aperture Radar Atlas) [3][4] is a web-based retrieval mechanism that has been online for over a year at the University of Lecce in Italy, at Caltech in California, and at SDSC in San Diego, California. The opening page presents a user with a picture of the Earth, and clicking on it can zoom in or out, depending on how the rest of the form is set. More distant views show coasts, countries, and rivers; closer views also show roads, railways, and city names. Also shown on the map are rectangular 'tracks' indicating that a SAR image is associated with that region. The user may select a particular track from the map by clicking on it; if multiple tracks contain the chosen point then the user is asked to select from a list of those tracks.

When a track has been selected, a Java applet appears, with a thumbnail image of the track area (perhaps 300 pixels wide by 1000 high), together with other controls that allow selection of a subset of the full track, an output format, and the false-coloring information. Since there are several SARA data servers around the world, the user can also select which of these should be used to create and retrieve the data. When the data-order is complete, the user can download the resulting image.

The SARA architecture is based on stateless web services (CGI scripts) that respond to a GET request with an HTML or plain text response. A GET request is a collection of keywords and values encoded into a URL, for example in the SARA application this URL:

```
www.cacr.caltech.edu/cgi-bin/sara_metadata_server?lon=-1&lat=52
```

returns some metadata about which track outlines contain the point of the given longitude/latitude point on the surface of the Earth. The request, as are all SARA requests, is in the form of keyword/value pairs, separated by the "=" character, with the pairs separated by the "&" symbol. The response to this request is a text file giving some metadata for the tracks that are found. Other SARA CGI scripts return HTML documents that may contain forms, images and applets. The SARA architecture consists of components with well-defined interfaces, so that components can be replaced easily.

2.2. Deficiencies of the SARA System

The SARA system provides an interface to a library of data objects, and it allows users to define exactly the subset of the data that they want, and retrieve it from any server that has it. However, in contemplating an extension to a system that can handle complex, supervised processing and data-mining, the existing architecture seems deficient.

First, all services are stateless CGI scripts, so that each request stands alone, rather than in a context of previous requests. Second, we feel that the interaction language is not sufficiently rich

to scale to a more complex system. The request to the script is always in the form of the keyword-value pairs that the HTTP-GET protocol allows, which can be somewhat restrictive. For example in implementing the idea of "an area on the surface of the Earth", we are restricted to rectangles specified by latitude/longitude limits. Extending to something like a polygon, with a variable number of points, or to a political region like a country, would be difficult to implement in this protocol.

There are deficiencies on the response protocol as well as with requests. The output from a CGI script is a MIME-typed data object; while it is convenient for passing arbitrary objects back to the requester, there are some problems. First, it is difficult for the script to create multiple data objects in response to a request, and secondly, the output cannot be flushed except when the whole data object is finished. A third problem is the lack of a separate diagnostic stream: before the script outputs data, it provides the MIME type that is coming; if there is then an error in the processing, it is difficult to return it to the user. The result is a mysterious, opaque failure, that could have been remedied or reported if the user knew the problem.

3. On-Demand Processing

3.1. Digital Puglia Project

The kind of active library that we are building is useful because often it is not sufficient to simply retrieve data to the client's workstation. Perhaps the data object is too large for downloading, or the client may not have the relevant processing software to reduce the data. The processing at the server may be as simple as a change of format, or it may be that a user does compute-intensive image processing such as principal component analysis, supervised classification, or pattern matching. By the term "procession on demand", we mean a data archive connected to a powerful compute server at high bandwidth, controlled by a client who may be connected at low bandwidth.

Processing on demand may also be necessary to get timely access to recent data; relevant to a hazard situation such as fire or earthquake, or to monitor a potential hazard such as an oil slick. In the case of on-demand SAR processing, the user interface should allow control of doppler parameters, chirp generation methods, and other parameters that control the processing software.

Another use for the active library is that there may be potential knowledge to be gained through combining and fusing data objects. In the Digital Puglia project, we will concentrate on multitemporal imaging, where the color channels of an image represent the surface of the Earth at different times, rather than different wavelengths or polarizations. In general, the collection of (monochrome) images from different times will not be geo-referenced: thus this is the major computational task. Once we have an infrastructure for creating multitemporal images, it is a short step to creating interferometric images, showing ground motion at the centimeter level.

3.2. Sessions and State

One of the problems with the existing SARA architecture is lack of state for the servers. Suppose for example that we have a service that creates maps of the Earth; the input is about the area that the map is to cover, the map projection, the types of features to be shown, what the output

format should be, and so on. Once the map has been returned, or perhaps a URL where the map image can be found, then the next thing the requestor wants is to evaluate the map projection: what point on the surface of the Earth corresponds to this point on the map. If the map server has state, then it can remember the last map it created, and evaluate the corresponding projection efficiently.

More generally, processing on demand that takes more than a few seconds to execute requires services that have state, so that a user can be authenticated, resources can be allocated and the job started. Enquiries to the service respond with status reports, and the service remembers where data products have been put.

As soon as we have a server which is accessible to many users, then we must be sure to separate the state of one user from that of another. This introduces the concept of a session, meaning that the server knows which of his clients is making a request, and the response depends also on the state associated with that user. Sessions may be implemented on a connectionless protocol (such as HTTP) if the user keeps a session ID, for example in a "cookie" stored on the user's machine.

Another complication that arises when servers have state is the assignment of memory or disk space in which the state is kept, and deleting state that is associated with defunct sessions. In our scheme, session management and authentication will be carried out by a suitable web server, such as the Java Web Server from Sun.

3.3. Why is XML Useful?

In a collaborative, scalable library project, XML has several advantages as the internal communication language. At the lowest level, XML does not require any software beyond a simple text editor, and should be readable with no special tools. In the next year or so, a large number of XML tools will enter the market and the shareware archives, allowing easy viewing, editing, and printing of XML documents. This means that when an XML writer is not communicating effectively with the corresponding reader, then it is straightforward to record and analyze the exchange between them, and thereby assign responsibility for the failing.

Superficially, an XML document looks like HTML, with 'tags' represented by angle brackets; some examples may be found later in this paper. The difference is that in XML the syntax – the set of tags and the way in which they are used -- is not fixed, as it is in HTML, but can be locally defined and extended. Let us consider a simple example of an XML document, one which describes a SAR track from the 1994/95 SIRC SAR catalog. Each element begins with a start-tag (e.g. <name>) and ends with an end-tag (e.g. </name>). The first few elements are no more that keyword-value entries, but the <area> element expresses an area on the surface of the Earth, where the track covers; currently only quadrilateral areas are supported, but it is easy to extend the language with other ways of expressing areas, for example circles, or countries. There are a variable number of <server> elements showing where the data for this track is mirrored, together with a collection of keyword-value pairs showing the files associated with this track.

The XML syntax is simple (people recognize its similarity to HTML), and it is powerful (extensible, hierarchical). Furthermore, it satisfies one of the requirements for flexibility that were stated in above: syntax checking. If XML is read by a so-called validating parser, then a

```
<track id=13242>
   <name>Almaz, Russia</name>
   <date>04-16-1994</date>
   <width>2824</width>
   <height>8000</height>
   <channels>8</channels>
   <area>
      <quadrilateral>
         <lon>47.425</lon><lat>45.669</lat>
         <lon>47.703</lon><lat>45.418</lat>
         <lon>48.719</lon><lat>45.963</lat>
         <lon>48.444</lon><lat>46.217</lat>
      </quadrilateral>
   </area>
   <server name=Lecce>
      <baseurl>http://sara.unile.it/.....</baseurl>
      <timeout>30</timeout>
   </server>
   <server name=CACR_HPSS>
      <baseurl>http://hpss.cacr.caltech.edu/.....</baseurl>
      <timeout>600</timeout>
   </server>
   <channel name=LHH>
      <file>pr13242_byt_hh</file>
   </channel>
   <channel name=LVV>
      <file>pr13242_byt_vv</file>
   </channel>
</track>
```

Coverage of the track on the surface of the Earth

Mirror servers that can deliver this track

The file names associated with the track

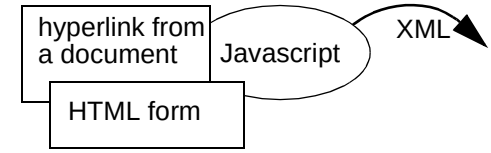
Figure 1: An XML file describing a data object from a collection of remote-sensing images. The location (URL) of a data object is obtained by concatenating the <baseurl> and <file> tags.

subsidiary document is also read that specifies acceptable syntax. This is a Document Type Definition (DTD) file. Each tag is in the document is checked against the DTD, that tags are properly nested, and that certain tags are present or unique. For the example of Fig 1, it is the DTD (not shown) which specifies that a valid <track> element must have <width> and <height> elements, that there may me more than one <server> element, and that a <timeout> element can only appear inside a <server> element.

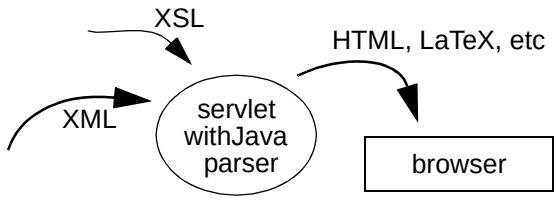
Thus we have a clear mechanism for collaboration: a group of developers agree on the syntax for a custom markup language, then write this knowledge in a DTD. Individuals can then create services and other processes that read and write documents in the custom language, and disputes can be settled by appealing to a validating parser. The committee writes the interfaces, the individuals write the compliant code.

3.4. Requests and Responses

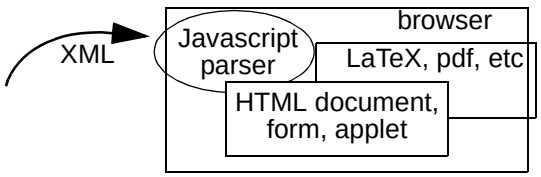
Figure 2 shows how XML can be created and consumed by a browser that is operated by a human and Figure 3 shows how XML can be created and consumed by a computer running a program. Figure 4 shows some tools that blur the distinction between a document and a program by creating an XML router. We can add <route> tags to an XML stream, with the idea that



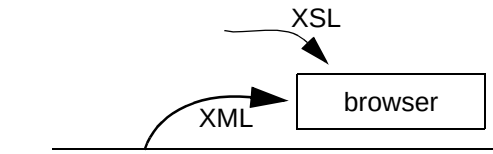
XML can be created directly from a web page or form through a Javascript parser embedded in the page.



XML can be processed on the server, with the addition of an XSL style sheet, to create human-readable documents or forms.

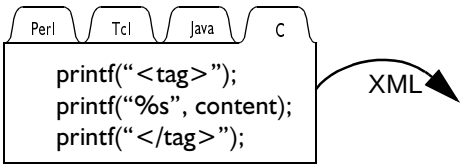


XML can also be processed on the client side with Javascript.

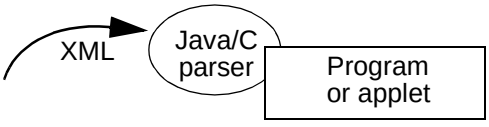


By mid-late 1999, the common browsers will display XML directly

Figure 2: Creation and consumption of XML by a browser. It can be created with a simple Javascript program that preprocesses form output before submission, or of course it can be typed in directly. In the next year or so, XML parsers and presentation will be integrated directly into web browsers, but until then, and for custom presentation, it can be parsed into HTML or any other presentation format either before it leaves the server or at the client. The presentation of an XML document is controlled by an XSL (eXtensible Style Language) document, which is itself an XML document.



Programs can create XML without any API, just from print statements



Programs can interpret XML with a parser in Java, C, Tcl or other language. The program is designed as an event-loop responding to tags and the contained text.

Figure 3: Creation and consumption of XML by a program. Creation is the simplest, involving nothing more than print statements. The data between the tags may come from a database, or it may be a serialization of a object in the program. The program can read an XML stream with a parser. The programmer provides a number of handler functions that are called when start-tags, text, end-tags, attributes, etc. are found in the XML stream.

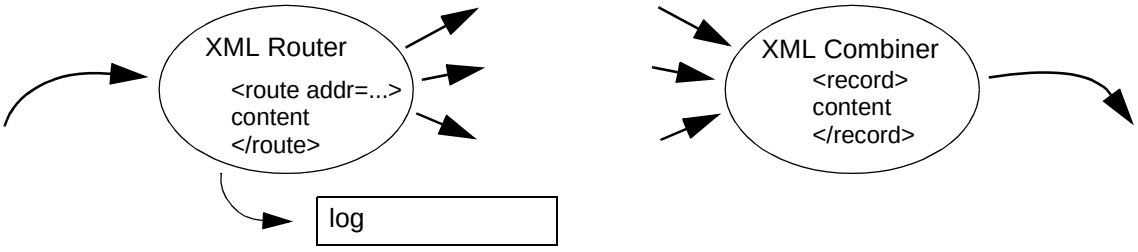


Figure 4: Routing and combining XML documents. In the router, a special tag is used to indicate content that is sent to the standard input of another service or applet. The addressing information is an attribute of the <route> tag. If not explicitly routed, content is simply logged. The combiner can be used to create log files. Input is copied to output until the </record> end-tag is found, then the next <record> tag from any input starts copying to the output.

different parts of the stream should be routed to different applets or servlets, and anything that is not explicitly routed is simply logged. In a similar fashion, a process that combines XML streams may be used to create a log file by concatenating <record> elements.

3.5. Services

In this paper, we shall define a *service* to be an executable program associated with a web-server that responds to an XML request with an XML response; where the markup languages (the DTDs) of the requests and responses are well defined. Further, the service is assumed to have a state which is different for each client, so that the service responds in the context of previous interactions with that client.

Restricting the output of a service to be only text-based XML means that other types of data output, such as images or other binary files, will be written elsewhere by the service, and the associated URL passed back to the client as part of the response.

In Figure 5, we combine the services discussed above into an active digital library, loosely based on the architecture outlined in [4] and [5]. A user begins a session with a manager service, which can then create or pass requests to other services. There are three services that act as brokers to a commercial database, to a commercial GIS product, and to a text-based legacy application.

The catalog service provides access to the catalog of data that is available from one of the geographically-separated Data Archives: a request specifies a subset of the tracks in the library by geography, date, satellite, or other criteria, and the response is a collection of track metadata as in Figure 1. The map service, attached to the GIS, is responsible for making maps: given the geographical area of the map, a desired projection, what to put on the map (roads, rivers, etc.), and the desired output format, the service creates the map object and writes it to disk where a web server can see it, then sends the URL back as a response. The integration service uses these two services to create a map with tracks from the catalog overlaid. Another service shows how our architecture can be used to provide web-based access to legacy applications, such as a parallel SAR processor, as explained in the next section.

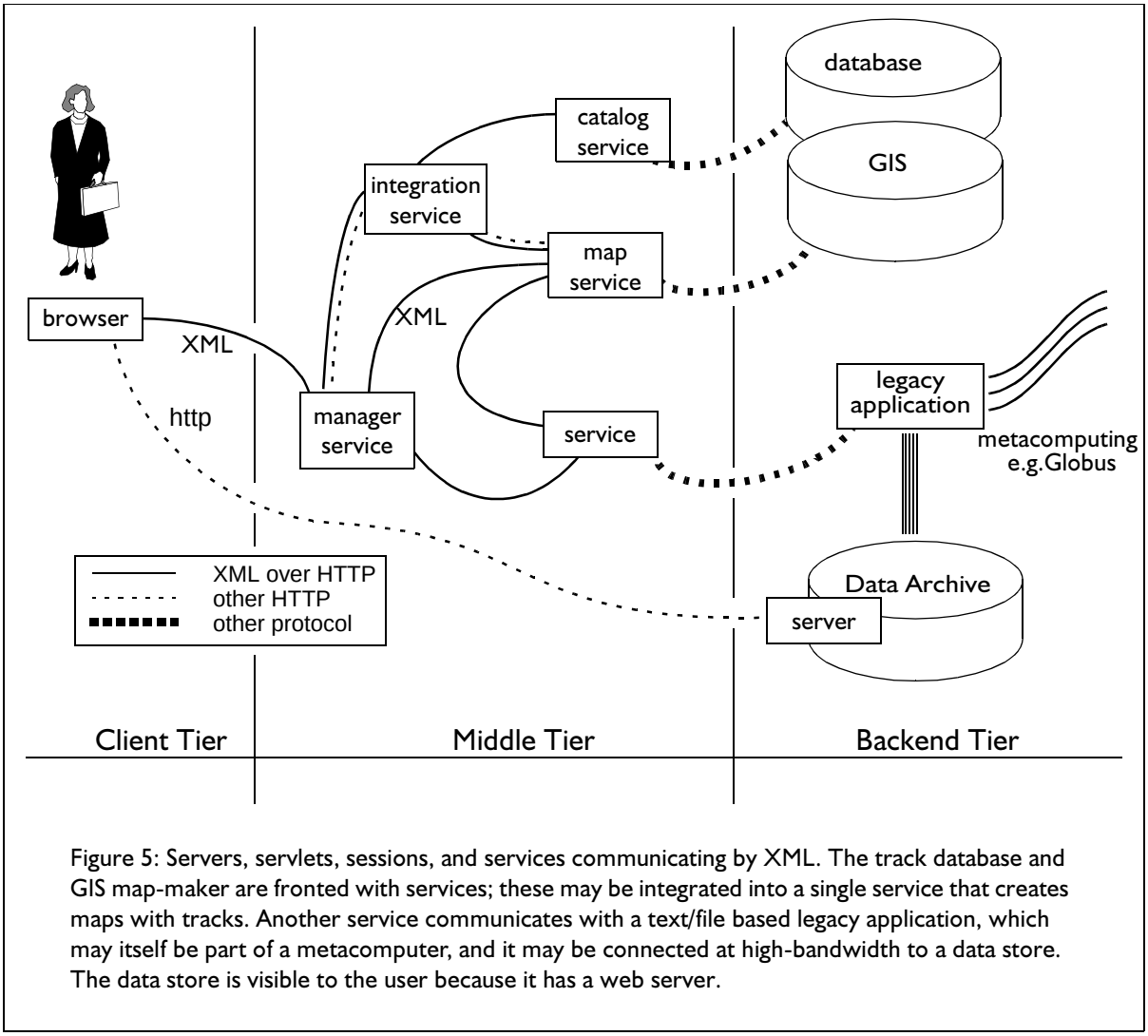


Figure 5: Servers, servlets, sessions, and services communicating by XML. The track database and GIS map-maker are fronted with services; these may be integrated into a single service that creates maps with tracks. Another service communicates with a text/file based legacy application, which may itself be part of a metacomputer, and it may be connected at high-bandwidth to a data store. The data store is visible to the user because it has a web server.

3.6. Three-Tier Architecture

Figure 5 shows how our XML architecture is divided into three tiers, the client, middle, and backend tiers.

The backend provides traditional, or "legacy" systems, which communicate using their own custom protocols: an example might be a vendor-specific database protocol, or a parallel SAR-processing application responding to an idiosyncratic set of commands and input files.

The middle tier provides a rich and extensible set of services for integration and management, for translation and brokerage. A broker might, for example, take data requests and make a judgement about which of a set of servers can provide the requested data; or which server can most efficiently provide the data. A translator might take a complex command directed to a legacy application, and convert it to a sequence of simpler commands that the application can understand, thus extending the legacy language. An integration service could make a request to a database, then use the results to generate queries or commands to another application.

The client layer is responsible for interacting with a human user and generating queries to the middle-tier services. The client tier may be a thin client, with nothing more than a web browser, or the browser may be enhanced with transparently-downloaded software components (applets). There may be plug-in components, or the client may have gone through a complex installation process to make a browser-independent interface.

4. Applications

To check the functionality of the XML architecture previously described, a joint project with the Italian Space Agency (ASI) is in progress. The Italian Space Agency (ASI) is a government agency, founded in 1988, having the responsibility to promote, coordinate and manage national programs and bilateral and multilateral cooperation programs; to promote and support Italian scientific and industrial participation in the European Space Agency (ESA) programs, in harmonization with national programs. One of the operational centers of the Italian Space Agency is the Center of Spatial Geodesy "Giuseppe Colombo" (CGS), in Matera, for Space Geodesy, Remote Sensing and Space Robotics. The CGS hosts the I-PAF (Italian Processing and Archiving Facility), a multimission facility for archiving, processing and distributing remote sensing data.

The XML-based SARA architecture previously described will be directed to providing automatically and on-demand some of the functions of the ASI I-PAF. The first step in designing an automatic facility for remote sensing is a careful analysis of the types of products usually required and distributed by the center, which will strongly influence the design strategies. These products can be divided in two broad classes: the so-called “standard products”, generally required by the application users, and the so-called “non-standard products”, mostly dedicated to research users.

The main characteristic of the “standard products” is that the processing parameters (i.e. the parameters needed by the SAR processor to produce these products) are fixed and established by international agreements. This means that to start the job, the SAR processor just needs to know what “kind of product” is required, the “raw-data” related to the selected area and the so-called “support data” (orbital data, calibration data, adjustment data, DEM-digital elevation model,...) which can be provided by the same operative center or from external coordinated processing centers. Usual “standard products” delivered by an operative SAR processing center are the PRI (PReclsion product), the GEC (Geocoded by a mathematical model of the terrestrial ellipsoid) and the GTC (Geocoded by DEM).

The “support data” depend on the required product. In general, the orbital and adjustment data are contained in the raw-data header, but if a Precision product (PRI) is required, then other information would be needed to start the SAR processing. This information might be the precision orbital data provided by specialized center, like the DLR-Germany or JPL-USA. Clearly, if the processing must be done in real-time, only the available support data will be used to produce low-resolution products, like the “survey X-SAR” and in general what are usually called the “quick-look” image.

In the case of GEC products, a precise mathematical model of the terrestrial ellipsoid is needed as “support data”, while for GTC products the precision of the Digital Elevation Model (DEM) strongly influences the quality of the final results. However, Digital Elevation Models, which are

usually created at the same processing center, can be also achieved by specialized centers (in Italy by the IGMI-Istituto Geografico Militare Italiano).

The "non-standard products" are, on the contrary, experimental products with low quality reliability and they are usually required by research users. In these cases, the processing parameters and the "support data" are continuously changed by the researcher for each trial. Moreover, it is mandatory to decide in advance, depending on the specific application user, what kind of information is needed. In fact, the strong interaction existing between the product to be delivered and the application user means that, in specific cases, the processed SAR image contains redundant information which can be avoided in the final product.

4.1. Creating Multitemporal SAR Images

An application that we shall concentrate on is the production of multitemporal images — the image demonstrates change over time. If the red, green, and blue components of the image correspond to different times, then gray means no change, and color corresponds to change. For agriculture and ecology we can see how plant growth is affected by seasons; for studies of environmental pollution, we can see danger areas and the effects of remedial efforts.

Creating multitemporal images requires a catalog service that can find the relevant images from different times, together with a supervised or unsupervised processing service that can register the images with each other, adjust contrast and brightness, and create the final annotated image. A GIS service can provide an overlay with orientation information such as roads, feature names, or latitude/longitude grid.

Once the infrastucture is in place for creating multitemporal images, it is a short step to a more powerful kind of processing: Interferometric SAR processing. By using phase information as well as amplitude from the returning radar pulses, this computational technology can show ground-movement at the centimeter level. Such effects may result from seismic or volcanic processes, from aquifer depletion, or motion of ice-sheets or glaciers.

Much of the processing for creating these images is quite intensive, and may require the use of remote supercomputers. To allow our system to securely connect with such systems, we intend to utilize the Globus metacomputing system [6].

4.2. Real-time Processing

We feel that the architecture outlined above would be well-suited for real-time processing. The newest satellites provide frequent overpasses of any given area: in the case of natural disasters, such as earthquakes, fires, or floods, it would be desirable to have rapid access to survey images such as SAR can provide. Real-time would be valuable when, for judicial reasons, an immediate monitoring is needed, to find individual responsibilities in the pollution of protected areas, illegal discharge areas, and so on.

In other applications, the requirement of real-time is not so important, for example, when land maps are periodically controlled to discover fiscal abuses or when public corporations have to upgrade land maps for territory planning and prevention. In the last case, more than the real-time, the proper integration of data of the same area coming from several sources is needed.

5. Conclusions

XML has gained acceptance for representing documents of many kinds largely because of its position as a compromise between the rigidity of HTML and the complexity of SGML. It remains to be seen, however, if XML can be used effectively for communicating objects in a real-world active digital library. Our objective is to discover the strengths and weaknesses of this approach.

6. References

[1] R. D. Williams, J. Bunn, R. Moore, and J. C. T. Pool, *Interfaces to Scientific Data Archives*, Report of a Workshop sponsored by the National Science Foundation, May 1998: http://www.cacr.caltech.edu/isda

[2] Jet Propulsion Laboratory, *The Imaging Radar Homepag*e: http://southport.jpl.nasa.gov/

[3] R. D. Williams, G. Aloisio, M. Cafaro, G. Kremenek, P. Messina, *SARA: The Synthetic Aperture Radar Atlas*: http://www.cacr.caltech.edu/sara/ and http://sara.unile.it/sara

[4] G. Aloisio, M. Cafaro, R. D. Williams, P. Messina, *A Distributed WEB-Based Metacomputing Environment*, Proc. HPCN Europe 1997, Vienna, to appear on Lecture Notes In Computer Science, Springer-Verlag. http://www.cacr.caltech.edu/~roy/papers/hpcn97.pdf

[5] R. D. Williams and B. Sears, *A High-Performance Active Digital Library*, Parallel Computing, Special issue on Metacomputing, November 1998 (to be published)

[6] I. Foster and C. Kesselman, *The Globus Project* http://www.globus.org