METADATA AND GEOSPATIAL DATA PROCESSING ON THE BASE OF XML AND GRID

Andrii Shelestov, Mykhailo Korbakov, Mykhaylo Zynovyev

Abstract: The software architecture and development consideration for open metadata extraction and processing framework are outlined. Special attention is paid to the aspects of reliability and fault tolerance. Grid infrastructure is shown as useful backend for general-purpose task.

Keywords: Metadata, ISO19115, Grid computing, Globus Toolkit, XML.

ACM Classification Keywords: D.1.3 Concurrent Programming – Distributed programming, D.2.3 Coding Tools and Techniques – Object-oriented programming, D.2.5 Testing and Debugging – Error handling and recovery, D.2.0 General – Standards.

Introduction

Metadata extraction, indexing and querying is important task from very different points of view. Consistent and actual metadata database enables effective use of data archives for users and create capabilities to develop new high-level services taking advantage of task run time prediction or automatic composition of semantic workflows. The current activities in metadata processing tools development are mainly targeting desktop indexing and search tools (Beagle, Tracker, Strigi, libferris [Martin, 2005]). However there are no available metadata processing systems that can handle geospatial data with their complex file formats, diverse metadata structure and complex queries. There is an ISO standard for geospatial metadata (described in UML) [ISO19115, 2003] as well as XML representation for it [ISO19139, 2007], but none of the available systems known to the authors take advantage of it. The approach described in this paper is targeting to fill this gap.

The work was supported by the INTAS-CNES-NSAU grant #06-1000024-9154 “Data Fusion Grid Infrastructure” and STCU-NASU grant #3872 “GRID technologies for environmental monitoring using satellite data”.

Objectives

The problem of extraction, storing and querying of metadata of geospatial data is very important in the context of development of distributed geoinformational systems of national or even larger scale. The large scale systems for environmental monitoring such as international GEOSS [GEOSS, 2005] or European GMES [GMES, 2005] operate on very large sets of data and need consistent and actual catalog of metadata to operate efficiently.

The development of system being described in this paper is carried within the number of national Ukrainian initiatives such as CosmoGIS and GEOUA and international grants INTAS-CNES-NSAU #06-1000024-9154 “Data Fusion Grid Infrastructure” and STCU-NASU #3872 “GRID technologies for environmental monitoring using satellite data”. These projects and initiatives are targeting on development and exploitation of Grid infrastructure in the tasks of geospatial data processing and environmental monitoring. The problems being solved in these systems involve satellite imaginary obtained via EUMETCast data dissemination system [EUMETCast, 2006], weather forecasts data and hydrological modeling. With growth of functionality of systems in development the need for metadata catalogue and queries processing engine became clean.

The Grid technology was found very suitable for development of such system being the basis for many data processing infrastructures and stated as a technological foundation for implementation of GMES system.

After analysis of requirement authors have identified a number of functions and features that must be supported by such system.

Functions:

1. To extract metadata from files of different formats. Scientific data often comes in quite unusual formats that require special software that can not be found on most of computers. Examples include XRIT envelopes that are used for distribution of MSG satellite or HRPT format that is used for NOAA satellite data. Data that

are using standard container formats like HDF of NetCDF are using different field structures. These factors cause the need to utilize external handlers to process different kinds of data.

2. To store the metadata in the queryable form in centralized storage. The common problem of storing abstract metadata in common relational database is large diversity of data structures that is causing denormalization of relations. The proposed XML-based solution of this problem is described in the following sections.
3. To provide interface to perform user queries. The user should be able to query metadata with different match criteria or their combinations. The proposed query language is W3C recommended XQuery language [XQuery, 2007].
4. To support geospatial queries. This requirement originates from geospatial nature of satellite imaginary. Minimal set of geospatial queries functions is support for windows queries, spatial join by intersection and different coordinate reference systems.

Features:

1. Distributed agent-server architecture. The data that require processing can be collected on different sites of data retrieval, processing and storing facilities. The metadata extraction module should operate closely to the data to reduce unnecessary data transfers. This leads us to separation of metadata extraction part that should be deployed on the storage side from the metadata indexing and processing part that should be deployed on the separate server.
2. Support for user extensions. The extraction of metadata should be performed by the means of external handlers that must process specific file formats and communicate with main extractor process via defined interface. The communication interface must be kept as simple as possible to avoid unnecessary limitations on implementation of external handlers.
3. Continuous processing. The metadata extractor storage-side agent must detect and process new data files at the moment when they arrive on the storage.
4. Fault tolerance. The metadata extractor must properly handle different kinds of OS-level errors like file access errors or network unavailability and also isolate errors in external file formats handlers including incorrect resource deallocation.
5. Access rights enforcement. Distributed system requires features of authentication and authorization to prevent abuse of its services by unauthorized users.

The proposed solution for implementation of these functions and attributes is given in the following sections.

Analysis of XML databases usefulness

The XML databases appeared as a natural consequence of rapid growth of XML as the standard for information exchange and increasing volumes of XML documents. As opposed to relational database in their current state, XML databases have very different functionality, performance and query processing capabilities thus making the right choice difficult.

XML database differs from a relational database in a number of directions.

- Relational database uses a row (relation) as the basic storage unit, while an XML database uses an XML document.
- Instead of using SQL for querying and updating data XML databases use the pair of XQuery and XUpdate languages.
- Relational databases are generally inefficient if the entire document is needed as it may be split across tables. XML databases may be inefficient if document or a part is requested in a form different from which it is stored. In this respect XML databases are similar to hierarchical databases.
- Relational databases suits best to the situations with simple enough data structures that can be described in the terms of relations without denormalization. XML databases can be used in situation with complex data structures that can not be easily mapped to relations.

Among many different XML databases we can identify two large classes [Srivastava, 2004]:

- XML-enabled Relational Databases. These databases are natural evolution of traditional relational databases with new features that allows developers to combine SQL and XQuery queries. The most

noticeable examples of this class are Oracle and MS SQL Server databases. The PostgreSQL database will implement XML features in close future.

- Native XML Databases. These databases were designed to store XML from the scratch. The researches are very active in this area so the databases differ greatly in sense of architecture, functionality, language support and other. There are much more representatives of this class comparing with XML-enabled relational databases. Most notable are Berkeley DB XML, eXist, Sedna, X-Hive, Xindice.

XML databases are naturally suited to storing geospatial metadata information for the reasons of already existing XML mapping of ISO19115 metadata standard and complexity of metadata structure. However the requirement for ability to process geospatial queries puts very strong restriction on the choices possible.

The only database engine available at the present moment with support for both XQuery and geospatial functions is Oracle 10g. The development of metadata indexing service uses Oracle 10g under the development license. PostgreSQL developers team claims to add XQuery features in the next releases. In this case PostgreSQL will become an engine of choice for its long-available geospatial features implemented in PostGIS extension (http://postgis.refractions.net/).

Proposed architecture

The system being described consists of two parts:

- The agent part, that deployed on the side of data storage and performs continuous monitoring of new and changing data files. The results of monitoring are pushed to the server part. The term “agent” was chosen with respect to SNMP (Simple Network Management Protocol) agents and backup agents terminology. The agent is a special kind of server’s client that only feeds data to it without or with very little of other kinds of interaction.
- The server part, that deployed on the side of metadata accepts metadata from multiple agents deployed on different hosts, stores retrieved metadata in persistent storage and handles user queries.

These parts perform very different functions and thus have different implementations. The server part is implemented as a service for Globus Toolkit 4 [Foster, 2005] container. Use of Grid approach for development of such service allows us to achieve transparent integration with other applied Grid services being developed in the scope of other projects and take advantage of consistent Grid middleware services such as authentication and authorization mechanisms, monitoring and event publication.

The following features of GT4 are used:

1. Security framework. GT4 allows service to use PKI (Public Key Infrastructure) with loosely coupled configuration of security policy down to the granularity of specific service methods. The security framework covers “Three A” problems: Authentication, Authorization, Audit in the standard way as well as providing means for channel encryption and integration with existing security infrastructures.
2. Index service. This standard GT4 service allows other services to publish information about their state and to query other services information. The server part of metadata processing system using Index service to publish information about last updates and overall availability.
3. Event publishing. GT4 allows services to publish events and subscribe to publication in the way, similar to Observer design pattern [Gamma, 1995]. This allows other services to monitor availability and updates of metadata database and to react to specific events. Event publishing reduces the load on application server by elimination the need for periodic checks.

The Figure 1 shows the high level overview of the architecture with several metadata extractor agents, metadata server, deployed into GT4 container and some users performing requests on it.

The server receives metadata information from remote agents and puts it into persistent storage implemented as XML database. The use of XML database to store ISO19139 data granules and other metadata in form of XML documents has a number of advantages comparing with traditional relational databases systems. User requests are sent to the server in form of XQuery language. It should be mentioned that XQuery isn’t very user-friendly and simple language so the optimal solution for user interface is high-level client application (either host-based or web-based) that will interpret user request in terms of ISO19115 and put it into XQuery.

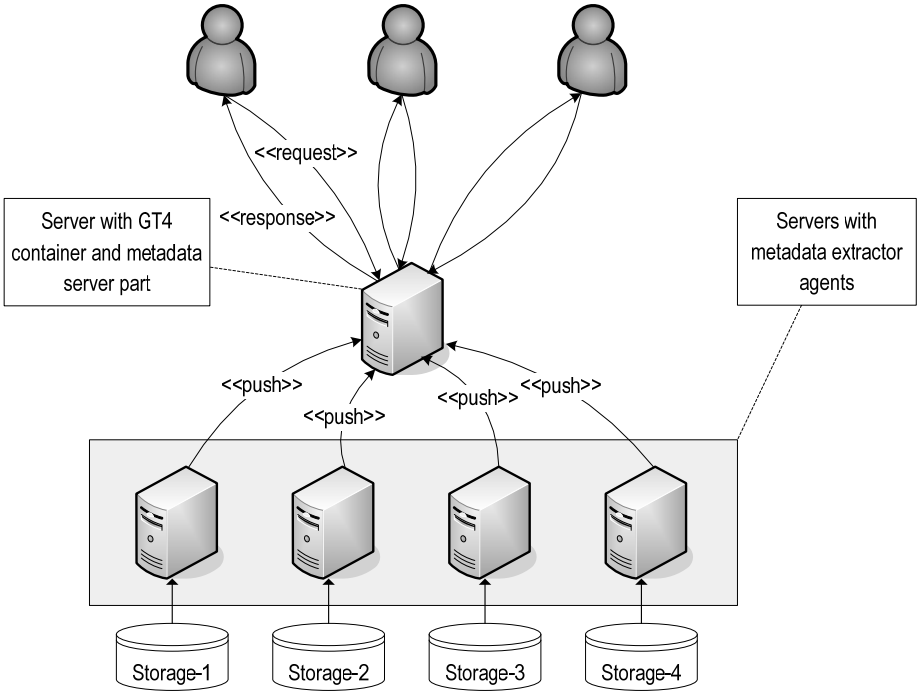

Figure 1. High-level overview of the architecture

To support clean shutdowns metadata extractor should support persistent storage for processing states of data The metadata extractor agent is implemented as a Python application for Linux environment and a set of extensions to handle specific file formats. The program uses FAM interface (File Alternation Monitor, http://oss.sgi.com/projects/fam/) to continuously monitor a number of directories specified by user, detects file format and then applies an appropriate handler.

files. At the present moment persistent storage is implemented by the means of the extended file system attributes that allows associating each file with a set of name-value pairs. Using of embedded database like SQLite or Embedded MySQL will improve compatibility of the software and will be implemented soon.

One of the design goals of the metadata extractor part was to avoid unnecessary limitations on implementation of external handlers. One of the possible ways to achieve it was to keep communication interface between the main program and extension as simple as possible. The current implementation allows every executable file put to special directory to be treated as an extension provided that it takes one command line parameter – the name of the file to process, returns valid XML document at the standard output and keeps the common agreements for the return code of a process. Such approach allows developer to choose most appropriate tool to develop a needed extension and shows zero learning curve.

The Figure 2 shows the class diagram of metadata extractor. Four classes implement the core functionality:

1. Extractor class is top of hierarchy and is responsible for startup of the program and construction of instances of other classes. The instance of this class is collecting information provided by DirectoryWatcher instances and sends it to server part.
2. DirectoryWatcher class is responsible for monitoring of a single directory in filesystem. The instance of this class starts FAM monitor and processes new files using FileHandler objects.
3. FileHandler class is encapsulation of external handler of file format. It responsible for running external process, retrieving the output and wrapping errors into exceptions.
4. EAFileJournal class is implementation of persistent storage for processing states of data files. It's implemented using extended attributes of file system.

Two composition relations marked as IPCObject represent wrapper objects that serve for interprocess communications implemented by the means of D-Bus protocol (http://www.freedesktop.org/wiki/Software/dbus) [Palmieri, 2005]. The details and reasoning of this design solution will be given further in "Reliability and Fault Tolerance considerations" section.

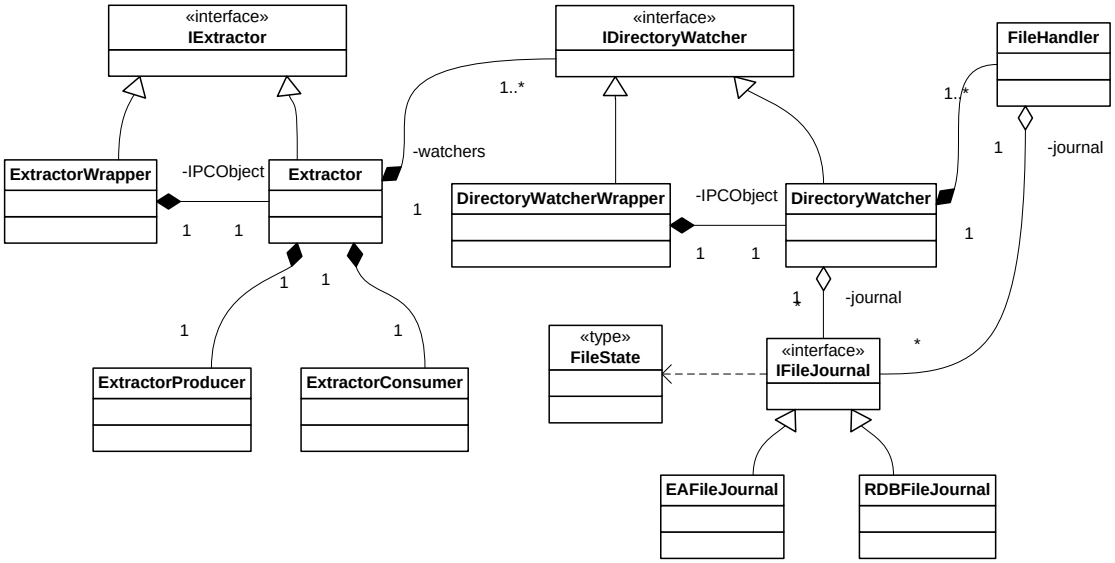

Figure 2. Class diagram of storage-side part of metadata extractor

Reliability and Fault Tolerance considerations

Reliability and fault tolerance were specifically stated in the objectives of the system. Analysis for potential faults has shown the number of sources (listed without any specific order).

- Disk read errors (SAN connectivity lost comes here too).
- Lost of network connectivity with indexing server.
- Erroneous input data.
- Exceeding of system resources (system memory is most probable candidate).
- Incorrect deallocation of resources leading to leaks.

Realization of potential fault will lead to the need of human interaction if the critical situation isn't handled in a proper way. To reduce the potential harm from these faults a number of special measures have been taken.

Disk read errors that can come either from physical hard drive fault or from lost of connectivity in storage area network do no require any special handling except proper detection of faulty situation and further periodical checks for disk availability. The same applies to the lost of network connectivity with indexing server.

Erroneous input data that is causing external file handler to crash with error should be marked as broken using persistent file state storage (FileJournal in terms of Figure 2). In those cases when input data consist of several files all of them should be declared as broken.

The exceeding of system resources and incorrect deallocation of resources is interdependent phenomena. The roots of the problem lies in the fact that the metadata extractor should run external code to retrieve metadata from different file formats. The following mechanisms were implemented to reduce the risk of crash for metadata extractor:

1. Isolation of error. Instead of running all of the tasks in a single process that can crash different tasks are running in separate processes. This applies both for external data format handlers and for different directories monitored by DirectoryWatcher instances (see Figure 2) running in separate process. The communication between external handlers and DirectoryWatcher instance is handled by standard output stream redirection that is very reliable. The communication between DirectoryWatcher instances and Extractor instance is performed by D-Bus interprocess message exchange system. In case of crash of process of external handler or DirectoryWatcher instance the operating system effectively frees all the system resources that crashed process have allocated. This doesn't apply to outer resources such as database connections but the use of such resource in described scenario quite unlikely.

2. Watchdog timer. The parent process of Extractor instance periodically checks the availability of underlying DirectoryWatcher processes via D-Bus message exchange. In case when DirectoryWatcher process doesn't reply in the specified time the process is killed and restarted by Extractor instance.

The last mechanism for supporting uninterruptible work of metadata extractor is logging system that records all relevant activities of it and can use quick contact methods such as email, IM or SMS to notify responsible person of all of system errors.

Conclusions

The authors have described the architecture of system for metadata extraction, indexing and processing targeting the geospatial data based on Globus Toolkit and XML databases technologies. The use of Globus Toolkit for such system shows a shift in the understanding of Grid systems. Authors believes that at the present moment Grid systems should be seen not only as a mechanism for supporting large computations and data transfers but also as a useful platform for value-added tasks.

Acknowledgments

This research is supported by INTAS-CNES-NSAU project "Data Fusion Grid Infrastructure", Ref. No 06-1000024-9154 and NASU Innovative Project "Development of pilot version of information infrastructure of Ukrainian segment of GEOSS".

Bibliography

[Martin, 2005] Filesystem Indexing with libferris. B. Martin. Linux Journal, 2005

[ISO19115, 2003] ISO standard ISO 19115:2003. Geographic information – Metadata

[ISO19139, 2007] Draft of ISO standard ISO 19139. Geographic information -- Metadata -- XML schema implementation

[GEOSS, 2005] Global Earth Observation System of Systems GEOSS. 10-Year Implementation Plan Reference Document. Noordwijk, Netherlands: ESA Publication Division. 2005. 212 p

[GMES, 2005] GMES: From Concept to Reality. European Commission. 2005

[EUMETCast, 2006] TD 15 – EUMETCast – EUMETSAT's Broadcast System for Environmental Data. Technical Description, (http://www.eumetsat.int/idcplg?IdcService=GET_FILE&dDocName=pdf_td15_eumetcast&RevisionSelectionMethod=LatestReleased), 2006

[Xquery, 2007] XQuery 1.0: An XML Query Language, W3C Recommendation 23 January 2007 (http://www.w3.org/TR/xquery/)

[Foster, 2005] Globus Toolkit Version 4: Software for Service-Oriented Systems. I. Foster. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[Welch, 2003] Security for Grid Services. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, 2003.

[Gamma, 1995] Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Addison-Wesley Professional, 1995.

[Palmieri, 2005] Get on D-BUS. John Palmieri. Red Hat Magazine, issue #3, January 2005.

[Srivastava, 2004] Comparison and Benchmarking of Native XML Databases. CS497 Report. Anand Vivek Srivastava. Department of Computer Science and Engineering, Indian Institute of Technology, 2004.

Authors' Information

Andriy Shelestov – Space Research Institute of NASU-NSAU, senior scientist; Ukraine, Kiev-03680, prosp. Glushkova-40; e-mail: inform@ikd.kiev.ua

Mykhailo Korbakov – Space Research Institute of NASU-NSAU, junior scientist; Ukraine, Kiev-03680, prosp. Glushkova-40; e-mail: rmihael@gmail.com

Mykhaylo Zynovyev – Bogolyubov Institute for Theoretical Physics of NASU, engineer, Ukraine, Kiev-03680, Metrolohichna str. 14-B; e-mail: Mykhaylo.Zynovyev@cern.ch