Access to Satellite Image Metadata on the Grid

Walter Daniel Lagrava Sandoval

Supervisor UNIGE: Prof. Bastien Chopard
Supervisors CERN: Dr. Massimo Lamanna, Dr. Birger Koblitz

Thesis submitted for the Degree of Master in Sciences at the University of Geneva

· 2006 ·
Abstract

The UNOSAT project is a United Nations programme that provides, among other services, satellite imagery to the international humanitarian community in case of natural disasters. UNOSAT has peaks of usage during catastrophes, when big quantities of computing power to process images are required, in addition to the storage space normally needed to stock the satellite images it owns. A solution to these necessities is offered by Computing Grids, distributed systems that can provide dynamic access to large amounts of computing and storage resources. This thesis is part of an ongoing effort to provide a UNOSAT service to deliver processed satellite images to field workers using technology and resources of the EGEE Grid. It covers the analysis of the requirements, design and validation of a metadata catalogue to access satellite image metadata on the Grid. This metadata catalogue has been used with two prototype services that I have developed, one to study the usage of the metadata by the users through a web-page, the other to study the access to the catalogue from mobile devices, and it is included in the UNOSAT web portal, which is a first implementation of a Grid service for UNOSAT, resulting from a collaboration between UNOSAT, NICE srl. from Italy and CERN.

Résumé

UNOSAT est un programme de l’Organisation des Nations Unies qui fournit, entre autres services, l’accès à des images satellites à la communauté humanitaire internationale en cas de catastrophe naturelle. UNOSAT a des pics d’utilisation pendant les sinistres, au cours desquels de grandes quantités de puissance de calcul sont requises, en plus de l’espace normalement nécessaire pour le stockage des images satellites. Les grilles de calcul, systèmes distribués qui fournissent un accès dynamique à une énorme quantité de ressources de calcul et de stockage, offrent une solution à de telles nécessités. Ce mémoire est une contribution à la création d’un service de distribution d’images satellites traitées aux travailleurs sur le terrain, utilisant la technologie et les ressources de la grille de l’EGEE. L’analyse des exigences, le design et la validation d’un catalogue contenant des métadonnées d’images satellites est expliqué en détail. Ce catalogue a été utilisé avec deux prototypes de services que j’ai développés, un service pour étudier l’utilisation des métadonnées via une page web, et un autre pour étudier l’accès au catalogue depuis un dispositif mobile. Le catalogue de métadonnées est aussi utilisé dans le portail web d’UNOSAT, première implantation d’un service sur la grille pour UNOSAT, et résultat d’un travail collaboratif entre UNOSAT, la compagnie italienne NICE srl. et le CERN.
# Contents

## Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Grid Computing</td>
<td>5</td>
</tr>
<tr>
<td>1.1 Historical Overview</td>
<td>5</td>
</tr>
<tr>
<td>1.2 The Grid Architecture</td>
<td>6</td>
</tr>
<tr>
<td>1.2.1 Virtual Organizations</td>
<td>7</td>
</tr>
<tr>
<td>1.2.2 Layered Grid Architecture</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Security on the Grid</td>
<td>10</td>
</tr>
<tr>
<td>1.3.1 Public-Key Cryptography</td>
<td>10</td>
</tr>
<tr>
<td>1.3.2 Public-key certificates</td>
<td>11</td>
</tr>
<tr>
<td>1.3.3 The X.509 Standard</td>
<td>11</td>
</tr>
<tr>
<td>1.3.4 The X.509 Proxy Certificate</td>
<td>12</td>
</tr>
</tbody>
</table>

## The EGEE Middleware Architecture

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Architecture Overview</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Data Services</td>
<td>14</td>
</tr>
<tr>
<td>2.2.1 Storage Element</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2 Catalogue Services</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Job Management Services</td>
<td>17</td>
</tr>
<tr>
<td>2.3.1 Workload Management Service</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2 Computing Element</td>
<td>18</td>
</tr>
</tbody>
</table>

## Metadata on the Grid

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 What Is Metadata?</td>
<td>19</td>
</tr>
<tr>
<td>3.2 The EGEE Metadata Interface</td>
<td>20</td>
</tr>
<tr>
<td>3.3 The AMGA Metadata Catalogue</td>
<td>22</td>
</tr>
<tr>
<td>3.3.1 AMGA Metadata Model</td>
<td>23</td>
</tr>
<tr>
<td>3.3.2 Replication</td>
<td>24</td>
</tr>
<tr>
<td>3.4 AMGA Use Cases</td>
<td>25</td>
</tr>
<tr>
<td>3.5 Other Metadata Middleware Services</td>
<td>26</td>
</tr>
</tbody>
</table>

## The UNOSAT Grid Project

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Analysis of Requirements</td>
<td>27</td>
</tr>
</tbody>
</table>
Introduction

On December 26th, 2004, the Indian Ocean tsunami, triggered by an earthquake of magnitude 9.1 with an epicentre close to the island of Sumatra, devastated the coastal regions around the Indian Ocean, killing 230,000 people in Indonesia, Thailand and along the north-western coast of Malaysia to thousands of kilometers away in Bangladesh, India, Sri Lanka, the Maldives, and even as far as in eastern Africa. Satellite images were crucial to assess the damage in remote areas where all lines of communication had been severed in order to coordinate the rescue effort and the widespread humanitarian aid offered by the international community. The UNOSAT project, a United Nations programme created to provide the international community and developing countries with enhanced access to satellite imagery, offered the first satellite image of the disaster area on December 29th. By January 14th, 2005, the online image bank hosted by the European Centre for Particle Physics (CERN), contained 100 processed and 670 raw satellite images of the region, downloaded 200,000 times by the end of January.

UNOSAT images are now used by the international community to organise aid and development and it is intended to make them available even to field workers using small mobile devices, for which these images must be suitably processed. Processing the high resolution images that UNOSAT provides, consumes a considerable amount of processing power. Images have a typical size of several hundreds of megabytes even in compressed form and must undergo complex image enhancement or compression algorithms before they can be delivered to the field workers. Because of the unpredictability of disasters, responding quickly to the needs in computing resources is difficult, as having these on standby would be uneconomical.

The advent of production Computing grids in recent years offers an elegant solution to these problems: Computing grids provide coherent access to large amounts of heterogeneous computing resources distributed across many participating institutions. They can quickly adjust to changes in the needs of computing power by changing the shares of their user communities. The UNOSAT grid project, a collaboration of UNOSAT, CERN and other partners has taken on the task to make satellite images and Geographic Information Services available to UNOSAT users worldwide using the EGEE (Enabling grids for E-sciencE) grid infrastructure.

The provisioning and processing of satellite images on a Computing grid need access to storage and computing resources provided by the grid as well as Metadata
services to look up appropriate images and to provide additional information on the image properties needed during the processing steps. This thesis describes the design and development of the metadata component of the UNOSAT grid service. To achieve this, first the requirements on the metadata of satellite images need to be studied and a suitable schema for this metadata needs to be designed, taking into account the complex structure of geographic data. Finally, APIs for the use in Graphical User Interfaces as well as for grid Jobs are designed and provided.

In the course of this thesis, two prototypes have been developed. The first allows to study the interaction of users when browsing image metadata to select images, providing information on which metadata is needed and how to present it to the users. The prototype works as a dynamic web page using AJAX programming techniques to provide quick feedback to the user. It includes also an application to administer the metadata information including the insertion of new entries (new satellite images with description). The second prototype focuses on accessing the satellite image metadata from mobile devices, taking into account long latencies and small bandwidth. It is implemented as a small Java application that can run on portable phones and other handheld devices. Several components of the prototype systems are now included in the UNOSAT grid Service.

The first chapter of this thesis will give an introduction to grid Computing, including a historical overview and in particular a thorough description of the security concepts on grids, which are typically using certificate based authentication models. The second chapter then provides a concrete description of the architecture of the EGEE grid Project’s middleware, in particular the storage resources and the job management services. The entire chapter 3 is devoted to metadata on grids and especially the AMGA metadata catalogue of the EGEE project, which is the main component used for the work of this thesis. Chapter 4 gives an overview of the UNOSAT grid project’s architecture while the requirements of the metadata component are gathered in the following chapter. The next two chapters describe the design and implementation of the two different prototype systems developed for this thesis. The last chapter shows how components of these prototypes are being integrated into the UNOSAT grid Service developed by UNOSAT, CERN and the Italian company NICE SRL.
Chapter 1

Grid Computing

The grid is an emerging computing model that aims to provide transparent access to computing power and data distributed over the globe. This collection of resources is accessible in a transparent way to users once access policies are met. The grid goes one step further than the World Wide Web in the sense that it does not only provide access to information stored in different geographic locations, but to several heterogeneous resources, such as computing power, data, software repositories or even sensors located all over the world.

The grid idea was developed by researchers with large requirements of CPU time and storage capacity to solve challenging problems. Nevertheless, with the evolution of the grid technologies, there is a growing interest for industrial and commercial applications as well.

The goal of this chapter is to present the main concepts of the grid computing model. I start giving an historical overview of the evolution of the grid. Next, I introduce the grid architecture that allows management of shared resources. The layered architecture of the grid is explained in order to show grid foundations. Finally, as security is always a major concern in public networks (the Internet being the support of the grid), I present the basics of grid security.

1.1 Historical Overview

The term “Grid” was first introduced by Ian Foster and Carl Kesselman in 1998 [2]. The name was proposed in analogy with the electrical power grid. In an electrical power grid, users plug electrical devices to it without caring about the origin of the power, and expecting to get the correct voltage. The grid intends to achieve a similar transparent access to computing power and resource sharing. In an ideal case, an user only would need to plug into the grid to access resources like computing power or storage.

Current grid technologies are the result of almost a decade of research in both, academic and industry fields. The grid technologies that manage the sharing of
resources are known as grid middleware. The name middleware is used because these technologies are an intermediate layer between physical resources and user applications.

In [3], the timeline of the grid development is divided in four important periods, which are:

**Custom Solutions.** In this phase starting in the early '90's, the term “metacomputing” appears. It is an effort to connect the power of several supercomputers in the United States. In order to achieve this, custom solutions based directly on Internet protocols are conceived. However, relying in these technologies limits security, scalability and robustness of the applications.

**Globus Toolkit.** Since 1997, the Globus toolkit is the *de facto* standard for grid technologies. The Globus Toolkit version 2 (GT2) is an open-source toolkit that defines and implements protocols, APIs and services used in several grid sites around the world. Even though the standards of GT2 are not subject to public review and are not formal, the fact that GT2 is the foundation for several grids allows some form of interoperation between them. For example we can cite the Open Science Grid (OSG) and LCG interoperability.

**Open Grid Services Architecture.** In 2002, the Open Grid Services Architecture (OGSA) is proposed, as a true community standard with many implementations, like the Globus Toolkit 3.0. OGSA is based on the Open Grid Services Infrastructure (OGSI), which describes the definition of interfaces to be implemented and their interaction. The OGSI framework allows users to define interoperable and portable services for the grid. OGSA is the first initiative to approach the grid and the industry in a more formal way by involving big companies like IBM to discuss the *fundamental structures*.

**Managed, Shared Virtual Systems.** Even though the OGSA approach marks an important advance for the grid technologies, there are still things that remain to be done in order to achieve the full grid vision and to arrive to solid generally-accepted standards. The future promises the arrival of new forms of virtualization, of sharing and of active management. Research in new computer science areas like peer-to-peer and knowledge-based systems will permit these developments.

### 1.2 The Grid Architecture

In this section, I introduce the Architecture of the grid. Given the importance of Virtual Organizations for the sharing of resources, I present this concept and then I describe the layered architecture of the grid in more detail.
1.2.1 Virtual Organizations

A Virtual Organization (VO) is a set of individuals and/or institutions sharing access to computers, data, and other resources [3]. This sharing should be controlled, with clear permissions and user policies. A VO is a sort of common denominator to manage users with common goals. From a management point of view, users belonging to the same VO have similar requirements and can be managed as a single entity. From a user point of view, resource sharing (especially data access) can be implemented in a natural way (e.g. all users in a VO share access to data, software usage, file catalogs, etc.). The VO concept defines how dynamic sharing should be done in a distributed system like the grid.

To show the heterogeneity of the VOs, we consider some examples from the EGEE community, that supports over 100 VOs. Each experiment at the LHC (the Large Hadron Collider at CERN) corresponds to a VO that includes physicists from all over the world. Their primary interest is to analyse the physics data to progress in their research. Almost 1,000 persons belong to the ATLAS experiment VO. The UNOSAT project’s VO of users who share access to satellite images for development and humanitarian purposes, consists of around 10 people.

Nevertheless, even if VOs can be so different in purpose, size and scope, they share a set of common concerns and technological requirements. There is a need for flexible sharing relationships, sophisticated and precise levels of control on the use of the shared resources and diverse usage modes, among others. Sharing between VOs is not restricted to data exchange, but it can include computer, software and other resources like sensors.

1.2.2 Layered Grid Architecture

As we can see in Fig. 1.1 the grid architecture is based on an “hourglass model” [3]. The goal of such an architecture is to allow many heterogeneous resources (Bottom of the hourglass) to be used by diverse applications (Top of the hourglass) with possible different scopes, by only using a small set of core abstractions and protocols (Narrow part of the hourglass). The wide parts (top and bottom) represent the diversity of different objects (applications and resources) in the layer.

In the following, the different layers of the grid architecture are briefly explained.

Fabric Layer

This layer consists of the shared resources to which access should be granted by the grid protocols in superior layers. Among these resources we can find computer clusters, supercomputers, storage systems, catalogs, sensors, etc. As an example, EGEE (the largest grid offering production-quality services) currently has over 200 resource centres all over the world.

The two most important mechanisms that resources should implement are introspection, the capacity to permit discovery of their structure, state and capabilities,
Figure 1.1: Layered Grid architecture (adapted from [3]). The figure sketches the diversity of resources (Fabric) that can be accessed by a big number of user applications (top of the sketch) via a small set of protocols (narrow part).

and resource management, to control the quality of service delivered. For example computational resources need mechanisms that allow the starting of programs, monitoring and control of execution of the processes. Management mechanisms are important to control the resources that are being used. On the other hand, introspection functions are necessary in order to determine characteristics such as the hardware configuration of the computing nodes, the operating system, etc.

Connectivity Protocols

The core communication and authentication protocols for grid network transactions are defined in this part. It contains two types of protocols: Communication protocols that allow the fabric layers to communicate and authentication protocols that are the basis to providing secure mechanisms to verify the identity of users and resources.

The communications needs are transport, routing and naming. The most used protocols are the ones from the TCP/IP stack, such as Internet (IP), transport (TCP, UDP) and applications (DNS, RSVP, etc.) layer.

The authentication mechanisms are based on protocols, which are specific to the needs of the grid. The Grid Security Interface proposes the uses of grid proxy certificates for authentication. Because of the importance of the security, this topic is developed more extensively later in this chapter.
As an example, we can cite the GEANT project and network\(^1\) interconnecting 26 National Research and Education Networks (NRENs) across Europe with a high-performance network.

**Resource Protocols**

Once the connection is established and authentication is performed using the protocols from the previous layer, the user needs a way to interact with the remote resources. It is the resource layer which has the role to provide such access to single resources.

This layer contains protocols for secure negotiation, monitoring, control, accounting and payment of sharing operations, based on the mechanisms provided by the connectivity layer. Protocols on this layer must capture the fundamental mechanisms to share across different resource types.

It is important to note that these protocols are the neck of the hourglass model (together with the connectivity protocols), so their number should, by definition, be limited. The reason for this is to avoid constraining the types or performance of higher-level protocols.

**Collective Layer**

This layer captures the protocols that are not associated with single resources, but are, instead, intended to capture interaction of a collection of resources.

As an example, implementations of this layer should provide services that allow the VO participants to discover the VO resources (Directory services). This is important to know which resources can be used, their load, status, etc. This service, on the EGEE architecture, is implemented by the Berkeley Database Information Index (BDII).

The system also needs services that permit the participants to allocate resources and schedule tasks on the appropriate resource, to monitor the resources of a VO (for failure, detect intrusions and overload), and services to optimize the performances of data access of the VO storage.

**Applications**

In top of the hourglass, we find the Applications layer that contains the applications built using the APIs for the precedent layers. This layer is quite complicated, as it can itself use elaborate libraries and frameworks to implement sophisticated algorithms for scientific applications.

During the EGEE User Forum held at CERN in March 2006, with over 200 participants, the different EGEE user communities presented their grid applications.

\(^1\)http://www.geant.net/
showing an amazing variety of applications, from particle physics to biology, nuclear fusion to archaeology.

1.3 Security on the Grid

Security is a fundamental in the grid implementation. Only authorized people should use the shared resources or have access to sensitive data according to defined policies. In this section I introduce the security that is currently used in Globus Toolkit. The grid Security Infrastructure (GSI)\(^2\) is the part of the Globus Toolkit that provides the fundamental security services. The security of this schema is based on the X.509 proxy certificates [4], which is an extension to the X.509 standard.

Proxy certificates are used instead of a login-password schema in order to avoid having a centralized authentication system in a distributed system like the grid. Proxy certificates also provide methods of credential delegation, which is necessary to minimize the times that the user must authenticate while making computations that require passing through several resources.

1.3.1 Public-Key Cryptography

In symmetric key algorithms, the user possesses a single key for encryption and decryption. In contrast, in Public-Key (PK) algorithms, the user has two type of keys, one for encryption and one for decryption. Furthermore, the encryption key, can be made public, so that anyone who knows it can encrypt a message for the user, but only the user with the right decryption key can decrypt the message. Often, the encryption key is called **public key** and the decryption key, **private key**.

Mathematically, the idea behind PK cryptography is to use problems such that encryption with the public key is computationally **easy**, but decryption without knowing the private key is computationally **hard**. This type of cryptography was invented by Whitfield Diffie and Martin Hellman in 1976. The most important algorithms based on this paradigm are RSA, ElGamal and Rabin, which are widely used in encryption and digital signatures [5].

RSA, invented by Ron Rivest, Adi Shamir and Leonard Adleman, bases its security on the difficulty of factoring big composite numbers. It must be said that it is the most widely used, simple to understand and to implement of the PK algorithms, it has resisted many years of cryptanalysis. ElGamal’s algorithm is based on the discrete logarithms problem. Rabin’s algorithm, on the other side, bases its security mechanism on the difficulty to compute the square roots modulo a composite number.

1.3.2 Public-key certificates

One important issue about PK cryptography is to know if a public key actually belongs to the person that we think it does. As an example, I present the man in the middle attack. Let us imagine that two people A and B wish to communicate using PK cryptography. An attacker, C, creates a private-public key pair, and achieves to make believe A that his public key is B’s. Whenever A sends a message to B, C can decrypt it, because it was encrypted with his public key, he reads it, and the he can encrypt it with B’s public key and send it, as if nothing had happened. In order to prevent this attack, PK certificates can be used.

A PK certificate is nothing more than the public key of a person signed by a trusted entity, which will be called Certification Authority (CA). In PK cryptography, a signature is the encryption of a document with one’s private key. Everyone can decrypt the message using the public key of the person who signed the document.

A certificate can contain more information than the public key, for example the name, address, and so on. The fact that a CA has signed it proves that this public key, as well as the additional information do belong to the person. The process that must be followed in order to get a CA signature is called vetting process and normally needs a truthful proof that the public key belongs to the person.

1.3.3 The X.509 Standard

This ISO authentication uses PK certificates. To generate the certificate, the trusted CA must, first of all, prove that a public key actually belongs to a person. The vetting process at CERN is be done by physically meeting the individual whose key will be signed.

The most important fields of a X.509 certificate are:

- Serial Number: is the unique name assigned to the person by the CA.
- Issuer: is the name of the CA.
- Period of time: is the duration of the certificate
- Subject: is the name of the person.
- Subject’s Public Key: contains the subject’s public key and the algorithm and parameters to generate it.
- Signature: is the CA’s signature, that is the information above encrypted with the private key of the CA.

If two persons want to communicate, they must verify the authenticity of their certificates. If the same CA issued both certificates, then there is no problem. Nevertheless, if the certificates were issued by different CA’s, then there is a tree-shaped hierarchy that needs to be explored, in order to find a common trusted CA for both of them.
1.3.4 The X.509 Proxy Certificate

X.509 proxy certificates are an extension to X.509 certificates, which are implemented by GSI. Proxy signatures allow person A to let person B sign as if he was A but without giving A’s secret key. Proxy certificates go further, as they intend to delegate more rights to the bearer than only authentication.

A X.509 proxy certificate serves to bind a public key to an individual as usual PK certificates. The format is the same as the X.509 certificates, allowing them to be used in protocols and in libraries as if they were normal ones. The main difference between a X.509 certificate and a proxy certificate is that the issuer (and signer) of the certificate are identified by a public key certificate or another proxy certificate instead of being signed by a CA. This permits to skip the heavy-weight vetting process to obtain a public key certificate from a CA.

In order to avoid misuse of proxy certificates, they must have a special section called Proxy Certificate Information (PCI) which tells the rights that have been delegated to the bearer of the proxy certificate, as well as to limit issuing of further proxy certificates with the current one. Special policy languages such as Keynote or XACML exist currently to define this policies.

The validation of a proxy certificate is performed in two steps. First, the validation of the certificate up to the X.509 public key certificate issued by a CA is performed. Then, the portion of the PCI is validated.

As proxy certificates are intended to have a short life span, the revocation of such certificates has not been a major concern in actual deployment, because the misuse of a proxy certificate is limited by its short time of validity.
Chapter 2

The EGEE Middleware Architecture

The EGEE grid middleware, called gLite, provides a concrete implementation of the software components to make a functional grid. This can be seen as a set of services implementing the basic functionality to allow the application to efficiently use the grid infrastructure. It is the combination of software components developed in the EDG project, and evolved further by the LHC Computing grid project, with reimplementation of several components done in the EGEE project itself. Components contributed by EGEE include the new Workload Management service and security based on a VO management system.

gLite has been designed according to a Service Oriented Architecture (SOA). A service is a well-defined component that is self-contained, presents a clean interface (with protocols) and does not depend on the context or state of other services. In a SOA the services provided are loosely coupled in order to increase the abstraction level for code re-use. Services communicate only through interfaces and protocols that are well-defined.

In this chapter a brief introduction to the EGEE grid middleware architecture [8] is given. To start, I will give an overview of the complete EGEE SOA architecture to show what services are provided in gLite. Because of the extent and complexity of the complete architecture, only data services and job management services will be described in more detail in later sections, because they are relevant for the UNOSAT grid project which will be presented in further chapters.

2.1 Architecture Overview

The following list gives an overview of the services provided by gLite:

Security Services encompass the Authentication, Authorization and Auditing services, which are used for identification of entities (users, systems and services),
management of access to resources and services and information about the security events. gLite uses the certificate-based authentication proposed by the grid Security Infrastructure (chapter 1).

**Information and Monitoring Services** allow publishing and consuming monitoring information about grid resources. More specialised services can be based on these basic ones, like job monitoring service, network performance monitoring, service discovery and others. In gLite, the Berkeley Database Information Index (BDII) provides information on resources and monitoring is done by the R-GMA (Relational grid Monitoring Architecture) service.

**Job Management Services** include Computing Elements (CE), the workload management, accounting, job provenance and package manager services. Communication between these services during a job request ensures job status consistency. The Resource Broker is used in gLite for workload scheduling and the submitted jobs are executed on the LCG CE.

**Data Services** are services for file lookup and data storage, which are implemented by the Storage Element (SE), Catalogue and Data Movement services. In practice, in the gLite middleware, the SE is charged of physically storing the files. The Storage Resource Manager (SRM) provides an interface to access the storage device and FTS (File Transfer Service) allows data movement between sites. The LCG File Catalogue is responsible of resolving the physical location of files. AMGA, which is presented in the next chapter, is responsible for metadata management in the EGEE middleware.

**Helper Services** are an addition to other services to provide a higher level of abstraction, better quality of service or better manageability of the whole infrastructure. Three such services are part of the EGEE architecture: Instrumentation Service (configuration of grid Services in a dynamic way), Bandwidth Allocation and Reservation Service (controlling and balance network usage) and the Agreement Service. This kind of services are still being implemented and will probably be available in future versions of gLite.

**Grid Access** is provided via CLIs and APIs of the above mentioned services. A machine enabled to access the grid is also called a User Interface (UI).

### 2.2 Data Services

The data services in the EGEE architecture are divided in three categories: storage, catalogues and movement. The lowest granularity of the EGEE data management services is often a file, so that data exchange is made via file exchange, in most of the cases.
2.2.1 Storage Element

Data storage resources are among the basic building blocks in a distributed computing infrastructure. Several storage resource types exist, ranging from a memory stick to a tape silo, each of them having different ways of accessing the data and with different Qualities of Service. The grid must provide interfaces for these hardware components in order that usage remains easy for normal users.

In addition, like any other resource, the access to storage must be controlled with well defined policies, the resource must be accessible according to the users’ needs and the usage monitored.

The EGEE middleware requires that each storage provider implements the Storage Resource Manager (SRM) interface, in order to not have to deal with the particular details of each storage device. The SRM is explained in detail in the web-pages of the grid Storage Management Working Group [14].

A Storage Element (SE) is a grid service that ensures storing and retrieving files from a data storage unit. The basic parts of an SE are:

- The storage back-end itself, this includes hardware and drivers.
- SRM service implementation for the given resource.
- A set of transfer protocols to move files between sites, which should include at least gridFTP [13].
- Native file access service in a POSIX-like I/O interface to access files locally on the site.
- Security, logging and accounting services.

SEs are supported by additional services like catalogues and reliable file transfer systems. In addition, client APIs are provided to read/store files in the SE.

The grid SE that is used by UNOSAT is Castor (CERN Advanced STORage Manager)\(^1\), a Mass Storage System (MSS) developed at CERN. Castor started working 8 years ago to provide storage for the experiments hosted by CERN with its own protocol. Since addition of the SRM interface, it can be used and accessed as a grid SE as well.

2.2.2 Catalogue Services

In the EGEE architecture, catalogue services hold information about the data stored on the grid. These catalogues manage the grid file namespaces and location of the files, storage and retrieval of metadata and information on access rights.

In this section I present the naming schema used to reference files in the EGEE architecture. Then I present the file catalogue proposed currently used in the gLite

\(^1\)http://castor.web.cern.ch/castor/
implementation: the LCG File Catalogue (LFC). Since access to metadata is the subject of this thesis the complete next chapter will be dedicated to the AMGA metadata catalogue of the EGEE middleware.

Data Naming

A user identifies a file stored on the grid with a Logical File Name (LFN) which offers a hierarchical namespace. The semantics are almost exactly the same as those in a Unix file system. However, this is not the only way to identify data on the grid. A file stored on the grid possesses also a Global Unique Identifier (GUID).

The LFN is the name of a file in a VO-internal logical namespace. LFNs are human readable, they must be unique inside a VO and they can be renamed (mutable). LFNs namespace is managed by the File Catalogue interface in the gLite implementation. A valid LFN might look like:

\texttt{lfn:/grid/unosat/images/2004/tsunami/India/image1.tiff}

In order to avoid clash of LFN between VOs, each of them has a separate Namespace.

A directory in the LFN namespace is manipulated just like in a normal file system. It can then be listed in the same manner and entries can be added, renamed or removed from it. In addition, the file catalogue allows \textbf{Symbolic Links} to be assigned to LFNs. These symbolic links work just like in the Unix file system. This implies that many symbolic links can be bound to the same LFN.

GUID (Global Unique Identifier) is a unique identifier just like the LFN, but is not human readable and is immutable. The uniqueness of the GUID is ensured by the UUID mechanism [10]. A GUID is a 36 bytes string. Users will deal with the human readable LFN instead of working with the GUID most of the time. However, it can happen that a grid file has only a GUID and no LFN associated, in which case users must use the GUID to manipulate the file. The immutability of the GUID is useful when referencing files.

Finally, the Storage URL (SURL) also called Physical File Name (PFN) identifies a replica stored inside an SE.

The LCG File Catalogue

When users and applications need to locate files (or replicas) on the grid, they use the File Catalogue service mapping between LFN, GUID and SURL of files. In the EGEE grid middleware this task is performed by the LFC (LCG File Catalogue). The LFC was conceived to overcome security and performance problems of an older system based on Globus RLS (Replica Location Service) [7]. The LFC implements several functionalities such as transactions, roll-backs, sessions, bulk queries and a hierarchical namespace for the LFNs.

For a file, the LFN is the main key in the LFC (Fig. 2.1), this also means that GUIDs cannot exist without being assigned to an LFN. Several symbolic links can
be assigned to the same LFN. Users can also add a simple metadata string to each LFN. The system attaches system metadata, GUID, file-size and access rights to every LFN.

The LFC can be used in a so called global mode to make the mapping between LFN and GUID or it can be used locally on a site to make the mapping of LFN/GUID to the SURL of a replica of a given file at that site.

2.3 Job Management Services

In grid terminology, tasks that request computation are usually known as jobs. The Job Management services provide the necessary framework to submit a job so that it is executed on the grid. The services provided are: Accounting, Computing Element (CE), Workload Management System (WMS) and Job Provenance (JP) service. In this section only the WMS and CE are presented in more detail, given the importance of both of them for jobs on the grid.

2.3.1 Workload Management Service

The Workload Management Service (WMS) is responsible for the distribution and management of tasks across grid resources so that jobs are executed in the minimal time, according to constrains imposed by the system (status of the resources), by the VO (available resource centres for a given user) and by the user (further selection
criteria). Nevertheless, the WMS might deal with other kind of tasks that use other kind of resources, notably storage and network.

The characteristics of the jobs are described in the Job Description Language (JDL) [12]. The JDL is used in all the WMS framework.

The core of the WMS is the Resource Broker (RB), which accepts and satisfies requests for job management from its clients.

A submission request consists of passing the responsibility for the job to the RB, which will select the appropriate CE for execution (matchmaking) taking into account the requirements described in JDL (user requests) and the availability of resources. After matchmaking, the WMS submits the job to a CE and takes care to control its execution on behalf of the user.

2.3.2 Computing Element

The Computing Element (CE) is a service that represents a computing resource. The main task of a CE is to execute a submitted job on a batch system. It must also provide other informations, such as its characteristics and current status.

A CE is a set of computing resources localized at a site (i.e. a computer cluster, a parallel computer) typically managed by a batch system. A CE can combine several heterogeneous hardware and software resources. It is therefore necessary that the batch system submits jobs to resources that are compatible with the requirements of the user. In this case the CE acts as entry point for a batch system (dedicated entirely) to grid users or having grid activity mixed with “local” users (users using the local batch facility).

It is important that a CE provides information about itself. First of all it should provide its characteristics, for example the types and number of existing resources, their hardware and software. Then it is necessary to get a status of the CE, such as the usage of the resources or the number of running and pending jobs. Finally, a CE needs to provide its policies on the resources like which users and/or VOs are authorized to run jobs on it, etc.

A CE acts as a generic interface to a cluster of Worker Nodes (WNs) that are the actual computing resources where the jobs are run.
Chapter 3

Metadata on the Grid

Metadata is vastly used in computer science to order and to describe data, as well as to improve retrieval of information. Given the important size and complexity of a distributed environment such as the grid, metadata is essential for most of the applications of the EGEE user communities. Metadata can, for example, be used to keep track of the millions of files that are spread over the many grid storages sites or to monitor jobs that are executed on the grid.

In this chapter I introduce AMGA (ARDA\textsuperscript{1} Metadata grid Application), which is part of the grid middleware and provides a service to organise, store and query metadata. In this chapter I explain what metadata is, to understand by a simple example how can it help in information retrieval. Then I describe the standard EGEE Metadata Interface, which is a proposal for metadata services that tries to capture the essential requirements of such services. I will then discuss the main features of the AMGA metadata catalogue, which was originally a prototype implementation of the interface, and now is widely used in many different EGEE user communities. Finally, in order to give a state-of-the-art of metadata services on the grid and to justify the choice of AMGA, several of these services are briefly presented and discussed.

3.1 What Is Metadata?

Metadata means ‘data about data’, that is data that describes some selected properties of another (usually bigger and more complicated) set of data. Metadata being smaller in size than the data it describes makes it possible to store it separately where it can be easily retrieved, such as a database.

As an example of the importance of metadata, let us consider a huge collection of files distributed over several storage sites in the order of hundreds of millions, as it is the case in the HEP experiments\textsuperscript{2}. Let us suppose that each of these files

\textsuperscript{1}A Realisation of Distributed Analysis for LHC group at CERN, http://cern.ch/arda

\textsuperscript{2}Using as guideline the figure of 10 PB of data collected each year and assuming 1 GB files, the total number of entry in a catalogue is of 10M (per year). The number of replicas should be of the
possesses simple metadata consisting of the name, emplacement and a description. As noted before, this metadata can be kept in a database. If a user needs to select data sharing certain common characteristics (e.g. HEP: data taken during March 2009 with a given detector setting, BioMed: brain images of persons of age between 20-29 years collected with a given technique, etc.) the metadata catalogue should return the list of file satisfying the user requirements.

On the other hand, if metadata was not available for our set, such requests cannot be easily fulfilled.

Outside the grid, metadata is used on several file systems to keep track of the contents of the disk volumes, such as HFS+ (Hierarchical File System Plus) of Apple Computer, NTFS (New Technology File System) from Microsoft or Linux/RK (Resource Kernel).

### 3.2 The EGEE Metadata Interface

The EGEE metadata interface [15] provides a generic metadata interface for metadata catalogue middleware services on the grid. This section gives an overview of the main components of this interface.

The interface proposes a generic set of basic operations for users. It also allows a broad range of implementations, given that it is not likely to have a single implementation that would satisfy the metadata needs of the whole range of grid applications. The reason for this being that the applications vary in size and access patterns in a significant way.

The concepts of the interface are **schemas**, **attributes** and **entries**. Schemas are collections of entries and may contain other schemas. A schema has a set of attributes of a certain type. Each entry, which represents the data item or resource being described, provides values to the attributes of the schema where it belongs. Schemas allow to give a logical structure to the metadata, organizing it in groups. They can be defined dynamically by the user.

In Fig. 7.1 an example of a schema is shown. It possesses \( N \) attributes, and there are \( M \) entries associated to it, each entry having a name and assigning a value to each of the attributes of the schema. As noted above, the example schema could itself contain another schema.

It is important to note that entries can not, by definition, exist outside a schema, and the only way that an entry can assign values to a set of attributes is by belonging to a schema. Nevertheless, it is an implementation choice to allow one entry to belong to only one schema, or to many schemas at the same time. The schemas can be organized in a flat namespace or in a hierarchy, depending on the implementation specifical needs.

The basic operations of the interface are:
Operations to create and delete schemas, in a dynamic way, according to the requirements.

Operations to add and remove attributes from a schema.

Methods to discover the attributes of a schema, because these can be defined dynamically by the user.

Operations to add and remove entries from a schema.

Finally, operations to list the schema(s) to which a particular entry belongs.

The design proposes a strategy to circumvent practical limitations to deal with database operations. First of all, the query language should be chosen depending on the back-end database. An SQL (Structured Query Language) based language would be a natural choice, as most users have a good knowledge of SQL, and also as the implementation can delegate most of the query work to the relational database management system. However, different implementations of SQL use different dialects. Another case is if an XML datastore is preferred, then the query language would be closer to XQuery. To avoid this, the interface does not define a query language, allowing the implementation to define it. A query, for the interface, consists of two text strings, one that contains the query itself, and another that gives the name of the query language used.

The other major difficulty of a query operation, is to manage large result sets. Due to memory requirements, reading all the results from the back-end in a single operation is not feasible for large result sets and several clients. To solve this, the interface proposes to use iterators to retrieve the results in small chunks. More specifically, the interface defines the method `query()`, to start the query and to get the first chunk, `nextQuery()`, to get the next set of results, and `abortQuery()`, to cancel a query. An opaque token is created after the call to `query()` and passed to the other two methods, in order to identify the queries.
query(), nextQuery() and abortQuery() methods can be implemented using either a stateful or a stateless model. In an SQL back-end, the stateful implementation can use cursors to read the results from the database. In this manner, efficiency and consistency are ensured, but the server must keep a state for each client between invocation. On the other hand, in the stateless case, the LIMIT clause can be used to return a specific number of results, using the token to store the current results read by the client, each call of the functions should pass an argument that repeats the query. A stateless service is less efficient (forces the clients to have multiple interaction with the server) and has consistency problems (if the database is updated between two queries).

3.3 The AMGA Metadata Catalogue

In this section, I introduce the AMGA\textsuperscript{3} metadata catalogue [16]. AMGA is a metadata service that implements the metadata interface presented above, allowing users to build flexible metadata schemas, populate the corresponding database and query them. AMGA started as an exploratory project to study metadata requirements for the LHC experiments (essentially to establish the interface that was described in the previous section). Since then, several user communities with different scopes, such as HEP or Biomedical applications, have started using it.

HEP applications use large numbers of files (10 millions or more) produced by the experiments. The corresponding metadata is known when the files are inserted into the system (e.g. simulation parameters used to generate the data). These files are usually read-only, once they have been produced. Each experiment is expected to register approximately 1 Petabyte of information ($10^6$ Gigabytes) per year, if we consider 1 Gigabyte files, then we can conclude that almost three of such files will be produced each second, so three entries will be inserted on the metadata catalogue per second. On the other hand, reading is harder to predict, given the number of people involved on the experiments all over the world (several hundred active users), with most probably some peaks of usage that should not create congestions.

As the users are spread on several geographical locations, there is a need for mechanisms to avoid high-latency connections. This can be achieved by using replication, more specifically read-only replicas. Concerning security, only authentication is required and it is not a primary issue for HEP. Nevertheless, malicious or inattentive deletion of data by non-privileged users must be avoided at all costs. In addition to file metadata, there are other structured metadata, such as calibrations or conditions for the experiment, that needs to be taken into account.

In Biomedical applications, the amount of data and metadata to be dealt with is smaller. In this case, the security is a major concern, given that metadata can contain sensitive information about patients and illness. This metadata must be handled very carefully. There is a need of fine-grained permissions on who can read

\textsuperscript{3}http://cern.ch/AMGA
the metadata of a patient. For this, a per-entry access permissions must be defined on the schema.

Each of the requirements for both of these EGEE user communities have been taken into account for the implementation of AMGA. In the following sections, there is a general overview of AMGA and a small explanation of the replication processes that are implemented on AMGA.

### 3.3.1 AMGA Metadata Model

AMGA uses a hierarchical file-system like model to structure metadata. In this case, a schema is represented by a directory, which can contain either entries or other schemas. Thanks to this, users can define a hierarchical structure which can help to better organize metadata. Concerning security, the implementation defines Access Control List (ACL) for a directory, and all entries contained in it share the same ACL. An entry can only be in single directory to avoid conflicts between security policies in different directories. AMGA can manage groups of users with different permissions on directories, for example only a root user would be allowed to add and remove entries, while all the other users would only be allowed to read the metadata. Authentication of users can be performed using either standard X509 certificates, grid-proxy certificates or login and password. SSL can be used to establish secure connections.

The choice of the back-end for this implementation are relational databases. Internally, each directory corresponds generally to a table, entries of a directory are rows of this table, and attributes correspond to columns. Directories can have an arbitrary number of attributes, which can be added or removed by adding or removing a column from the corresponding table. A master table contains the index of all the directories, as well as other properties for each directory. Since only two operations are required to access a directory (one to the index and one to the directory table) in most cases, this solution proves to be flexible and efficient.
In Fig. 3.2, the main components of the AMGA implementation are shown in more detail. The metadata server is implemented as a multithreaded C++ server, using a stateful model for queries, as seen on the previous section. In order to communicate with the server, the client opens a session where he authenticates if needed. Then he can send the queries to the server by using successive connections. The server can support several sessions at the same time and each session consumes resources. Due to this, malicious or buggy clients can use all the resources the server has. To avoid this, it automatically kills unused sessions for a long time and each client has a limit number of sessions on the server. If the stateless model would have been used, this problem would not exist, but as noticed before, there would be a performance penalty.

The server supports several storage systems by using modules. There are modules for most of the relational databases including PostgreSQL, Oracle, MySQL and SQLite. If no back-end is used, then there exists a stand-alone implementation which stores the metadata on the filesystem.

Concerning the front-end supports, there are two access protocols on the implementation, SOAP and TCP streaming. The SOAP protocol is based on the gSoap toolkit\(^4\). The TCP streaming front-end uses a text protocol similar to TELNET, so that commands and answers consist only of plain text messages. This is a deviation from the interface, since it should use message-based protocols. However, the commands on the streaming protocol have the same behavior as the operations which are defined on the EGEE interface. Using a streaming protocol also represents an advantage when dealing with large results, as they can be send at once as a byte stream to the client, while using a message-based protocol may need several round-trips to achieve this.

AMGA provides a command line interface, an interactive client and client libraries in C++, Java, Python and Perl.

### 3.3.2 Replication

By using replication, scalability, performance and fault-tolerance required to support geographically distributed sites can be achieved. The performance is greatly enhanced because replication can create read-only replicas of the distant files into a closer location, in this way the main server is not offloaded. On the other hand, if a server is out of order, the users can always read the files they need from some other site. The LCG 3D\(^5\) (Distributed Database Deployment) project provides solutions for replication in the HEP community, however for Oracle back-ends. An ongoing activity on replication with AMGA is on the other hand a subject of an ongoing Ph.D. thesis [17].

Replication in AMGA uses an asynchronous master-slave architecture. The master-slave architecture was chosen thinking on the target applications of AMGA,

\(^4\)http://sourceforge.net/projects/gsoap2/
\(^5\)https://twiki.cern.ch/twiki/bin/view/PSSGroup/LCG3DWiki
and avoiding the complexity of a multi-master system. The process is asynchronous because the synchronous replication does not suit well to Wide-Area Networks. There are two basic types of replication, full (Fig. 3.3) and partial (Fig. 3.4). In full replication, the slave gets the whole hierarchy of schemas, while in partial replication, only a selected subset of the schemas sent to the slave.

The master server possesses a log file of the changes created to its back-end. This log, which is sent to the slave, contains all the metadata commands that were performed to update the catalogue and some contextual information so that the commands can be replayed.

### 3.4 AMGA Use Cases

AMGA is being used in several projects that have metadata needs, either for evaluation or for production. The LHCb\(^6\) experiment has evaluated AMGA for their bookkeeping service, which controls the processing of the huge amount of data taken by the experiment and stored in files on the grid (about 10M so far). Ganga\(^7\), which is an user interface for job submission to the grid, developed by LHCb and ATLAS together with the EGEE/LCG project, uses AMGA to store metadata describing

---

\(^6\)http://lhcb.web.cern.ch/lhcb/

\(^7\)Ganga - Gaudi/Athena and grid Alliance, http://cern.ch/ganga/
the status of the jobs.

In the Biomedical community, the Medical Data Manager (MDM) application is being developed. MDM is a service that stores medical images on the grid, allowing users to retrieve them for post-processing or viewing. This application has some similarities with the UNOSAT application that will be presented in a later chapter.

3.5 Other Metadata Middleware Services

Because of the importance of metadata services for the usage of large scale local or wide area storage resources, many groups have made efforts to investigate and to implement such services.

Among the earliest metadata middleware we find the Metadata Catalogue Service (MCAT), which is part of the Storage Resource Broker (SRB) [18]. This project, developed by the San Diego Supercomputing Center, aims to provide an abstraction layer over heterogeneous storage devices and file systems either inside a computing center or across computing centers. MCAT stores the metadata hierarchically using a tree of collections. Later versions of MCAT [19] have replication and federation mechanisms. Globally the functionality of MCAT and AMGA are very similar, but MCAT is integrated in SRB and it only works with file metadata.

The Open grid Services Architecture (OGSA) [20] proposes a generic database interface developed by the Database Access and Integration Services (DAIS) group. The DAIS specification exposes the data storage to the user, instead of using a generic query language like proposed on the EGEE interface. This has the advantage to provide all the features of the back-end to users. On the other hand, the user is constrained to use the respective SQL-dialect of the back-end, which is a problem when working in an heterogeneous environment as the grid. The OGSA-DAI project implements the DAIS specification in the Globus Alliance’s middleware [21].

The Metadata Catalogue Services (MCS) [22], also developed by the Globus Alliance, uses the EGEE metadata interface. MCS is being developed on top of OGSA-DAI, it provides hierarchical organisation of the metadata, flexible schemas and hides the storage back-end to the user. The combination of OGSA-DAI and MCS provides much of the functionality of AMGA.

Several LHC experiments have implemented their own specific metadata catalogues using a standard relational back-ends and providing an intermediate layer to access the catalogue on a distributed environment. These catalogues are AT-LAS Metadata Interface (AMI) [23], RefDB [24] from the CMS experiment and the Alien Metadata catalogue [25] from ALICE experiments. The main issue with this catalogues is their scalability, a problem that has been addressed in AMGA.
Chapter 4

The UNOSAT Grid Project

The UNOSAT project\footnote{http://unosat.web.cern.ch/unosat/} is a United Nations programme that provides access to high quality satellite imagery and Geographic Information System (GIS) services to the international community. Satellite imagery and geographic information are intended to be used by the humanitarian community in case of disaster and for the planning of sustainable development. The images owned by UNOSAT are acquired from several commercial providers and have a significant value. UNOSAT has computing power and storage requirements given the number of images they own.

The UNOSAT consortium is formed by several private and public partners. CERN is UNOSAT partner since 2002 and hosts several Terabytes of satellite images of the UNOSAT project in its tape storage pool.

In this chapter I present the requirements, the intended workflow and proposed architecture of the initial implementation of a UNOSAT service based on grid components. Use cases describing the requirements for the application are explained first, in order to identify the main components, actors and workflow of a simple request. Then, I present a proposed architecture that captures the requirements of the UNOSAT grid project. The satellite image metadata catalogue, that I have conceived and is presented in the next chapter, is part of the proposed architecture.

4.1 Analysis of Requirements

Use cases are widely used in software development to capture the essential requirements of a project. In this section, I summarise the requirements of the UNOSAT grid service in form of two simple use cases.

During natural catastrophes and disasters, UNOSAT has peaks of usage because of the number of requests for images of the affected regions. In these periods, UNOSAT requires large amounts of computing power to process the needed satellite images as well as storage for the processed images. As people cannot predict when a natural disaster will happen, UNOSAT needs to have access to the computing power
when the situation requires it. By using the grid, UNOSAT can have this computer power when needed, as well as permanent storage space for the images.

The main use case corresponds to the activity of a field worker in a disaster area requesting a satellite image from this region. He uses a portable device such as a mobile telephone or a laptop computer to connect to the main system. The user must authenticate for example by using a simple method such as login and password. The system on the other side will process the input from the user and provide as a result the image that satisfies the search criteria. This image must be processed (at the very least compressed and cropped), so that it can be visualized from the mobile device of the field worker. The system uses applications that allow it to contact the grid. The time that the user can wait for an image ranges between 15 and 30 minutes.

Another use case of the system corresponds to the activity of a UNOSAT administrator that wants to upload an image to the grid, and insert the corresponding metadata into the metadata catalogue. It is important that only administrators are able to modify the metadata catalogue and upload images into the grid. In addition, it should also be possible to browse the image database to find the images that exist inside the catalogue. In order to ease this browsing, several selection options should be implemented, for example selection by keyword (i.e. country name, provider, etc.) or selection of a region in an interactive map.

In both use cases, users and administrators must be unaware of the underlying grid mechanisms that they will be using. In the case of the users it is mandatory, because, most of the time, users will be novice to the grid technologies. On the other hand, hiding the grid is a desirable feature in the case of the administrators, but not mandatory, since administrators will be trained computer engineers.

### 4.2 Proposed architecture

The architecture for the UNOSAT service is the result of a collaboration between CERN and UNOSAT to fulfill the use cases presented in the previous section. A sketch of the architecture is shown in Fig. 4.1. In order to hide the grid services to the users some kind of portal technology is necessary.

The main part of the architecture is contained inside the portal, which is used to shield the user from the authentication mechanisms of the grid that are technically too demanding to be implemented by mobile devices. This portal implements a lightweight authentication which maps into a grid proxy certificate creation for the user. The portal thus takes on the user’s role on the grid. The portal also implements grid data and job management in a transparent way. Within the portal an application based on the AMGA client API communicates with the AMGA server. This server contains the satellite metadata catalogue and allows to choose the image by the coordinates given by the user. Another application in the portal creates a job description and submits it to the grid. The job sent to the grid downloads an image
Figure 4.1: Sketch of the components of the proposed architecture for the UNOSAT grid service.

to the CE from an SE and processes it.

An administrator is able to use the portal to copy and register an image and insert the associated metadata to the AMGA catalogue as well. The portal will provide a different interface to an administrator in order to achieve this. The administrator must be able to browse the contents of the metadata catalogue, in order to know the images that are currently stored on the grid.

4.3 Workflow

I present the workflow for a request of a processed image to the portal in order to better explain how the components interact. The user must measure his current position in terms of longitude and latitude using a GPS device and authenticate to the portal before starting to use it.

In Fig. 4.2 the user submits his current coordinates to the portal. The portal
queries the AMGA metadata catalogue with the input from the user. The server answers with the LFN of the best image for the user. The LFC maps the LFN into the SURL, which defines the physical location of the image in the SE.

Using this SURL, a job that copies the image and processes it into a CE is submitted. The processing of the image consists mainly in cropping the image around the region of interest of the user and compressing it. After the job is finished, the portal retrieves the resulting image from the CE and sends it back to the user.

4.4 Discussion of the Architecture

The architecture presented in the previous section has many advantages. The workflow is easy to understand, it provides the needed features, and it is modular. This last advantage allows the division of work between developers, as long as communication standards are defined and used.
A variation of the architecture would be to add an intermediate application where more image metadata is provided to the user before submitting the job. Based on this metadata that could contain among other fields the date of creation of the picture and its resolution, the user could refine the selection further.

The basic processing of the images consists in cropping around the region of interest and compression, but it would be very useful to add other processing in the future. For example, it would be very interesting to create processed images on-the-fly out of raw images taken in different spectral bands using user-specific parameters or algorithms.

The usage of the portal allows to efficiently hide the details of the grid authentication mechanism from the users. However, it requires the server to store an un-encrypted copy of the user’s certificate. While the validity of a proxy certificate is usually restricted to some hours, within this period of time, a stolen certificate could be used to take on the user’s identity to contact any grid service. A solution to this problem would be to delegate a certificate to the portal that has restricted capabilities (i.e. it cannot be delegated again).
Chapter 5

Satellite Image Metadata Schema

Satellite image metadata is fundamental for the usage and understanding of the image it describes. Every image is delivered with a metadata file describing among other things the region where the picture was taken, the date and the wavelength used by the camera.

UNOSAT owns images taken by several different satellites. Satellites can be grouped by the resolution of images that they provide. For example, the Ikonos and Quick Bird satellites take high resolution images, Landsat 7 and Spot provide medium resolution images and satellites like Meris and Modis provide low resolution images. Other satellites provide radar images, for example: ERS, Radarsat and Formosat. Each of these images comes with associated metadata, in a provider specific format. For example Landsat 7 metadata comes in a plain text file where each line describes a property while Quick Bird metadata uses an XML format. Even though there are many different formats, they contain similar information. It is therefore possible to translate all these formats into a single agreed format.

In this chapter I present the metadata schema for satellite images that I have conceived. I start by providing details on the Geonetwork open-source project which is used by UNOSAT to store image metadata in a common XML format. Then I introduce the Geographical Information Systems that are currently used in relational database back-ends. In particular, I have measured the performance of spatial indexes for queries on geometric objects in PostgreSQL PostGIS\(^1\) extension. Finally, I describe in detail the satellite metadata schema for a grid-based metadata database.

5.1 Geonetwork

Geonetwork\(^2\) is a web geographic data and information manager. It is based on standards to improve the sharing and exchange of spatial databases that include maps, satellite images and related statistics in digital format. Geonetwork is an

\(^1\)http://postgis.refractions.net/support/wiki/
\(^2\)http://geonetwork.sourceforge.net
effort of several UN organizations and projects such as FAO (Food and Agriculture Organization of the United Nations) and the UNEP (United Nations Environment Programme).

Geonetwork possesses an internal Mckoi SQL database. Administrators are able to add images metadata in XML format to this database. The XML file is inserted in the database without extracting any information from it, because Geonetwork does not provide a dedicated schema for the metadata contained inside the XML files that could profit of the underlying database capabilities to improve information storage and retrieval. Geonetwork uses keywords with a keyword type for each metadata file, in order to provide users a way to effectively find metadata of images that verify some criteria (i.e. all images of Switzerland or all images taken by the Landsat satellite).

The mandatory fields of the Geonetwork metadata are specified in the Table 5.1.

Table 5.1: Mandatory fields of the Geonetwork metadata according to the ISO 19115 standard.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>title of the set</td>
</tr>
<tr>
<td>date</td>
<td>dates important for the image</td>
</tr>
<tr>
<td>date type</td>
<td>description of a date (e.g. publication, creation)</td>
</tr>
<tr>
<td>language</td>
<td>language in which the metadata is written</td>
</tr>
<tr>
<td>abstract</td>
<td>brief description of the image</td>
</tr>
<tr>
<td>topic categories</td>
<td>categories of the image (e.g. radar image, satellite of origin)</td>
</tr>
<tr>
<td>purpose</td>
<td>description of the usage of the image</td>
</tr>
<tr>
<td>denominator</td>
<td>scale equivalence with a hard copy of the map</td>
</tr>
<tr>
<td>update frequency</td>
<td>frequency of update of the picture (monthly, weekly, never, etc.)</td>
</tr>
<tr>
<td>metadata author</td>
<td>information of the metadata author (name, role, coordinates, etc.)</td>
</tr>
<tr>
<td>point of contact</td>
<td>same information as metadata author about the point of contact for the image</td>
</tr>
<tr>
<td>dimension</td>
<td>information about a dimension (name, size, etc.)</td>
</tr>
<tr>
<td>geographic box</td>
<td>border (north, south, west and east) coordinates of the image</td>
</tr>
<tr>
<td>character set</td>
<td>encoding of the metadata</td>
</tr>
<tr>
<td>supplementary info</td>
<td>other details of the image</td>
</tr>
<tr>
<td>keywords</td>
<td>keywords of the image to improve the queries to metadata</td>
</tr>
</tbody>
</table>

For the UNOSAT grid project, the Geonetwork metadata specification has been
adopted to represent the metadata. Therefore, all the metadata formats must be
translated to the Geonetwork format.

I have developed a Python program that converts the Landsat 7 metadata for
raw spectral bands, into Geonetwork metadata in a semiautomatic way in order to
have enough example sets of metadata to test the satellite image metadata schema
that I have proposed. The user must only provide the country and the place where
the image was taken. The rest of the fields are parsed from the Landsat 7 metadata
file. An XML file, that one can insert into Geonetwork, is produced for each of the
raw bands. In order to ease the process, an XML template defining all the fields
to be filled is used. Such semiautomatic programs to make the conversion between
metadata formats are necessary for all the satellites, because this avoids human
mistakes while copying the files and quickens the process.

5.2 Analysis of the Metadata Requirements

UNOSAT currently owns between 1,000 and 1,500 satellite images. This number
grows by an order of few hundred each year. The expected number of entries on the
metadata catalogue will have, when all the images will have been uploaded to the
grid, the same order of magnitude as the number of images.

Security is a concern to UNOSAT because of the commercial value of images (e.g.
a 60 km by 60 km archive image from the Spot satellite costs 2,300 Euros and 10 km
by 10 km high resolution images can go to 3,000 dollars [26]). Besides the images
themselves, there is no sensitive data stored in the metadata catalogue that needs
to be specially protected. This is a considerable simplification for the security of the
schema (medical image metadata for example contains sensitive information, some-
times more sensitive than the image itself). Therefore, basic proxy certificate based
authentication is sufficient for granting access to the catalogue. On the other hand,
it is important that accidental or malicious deletion or modifications of metadata
entries are avoided in order to preserve all the important information, so it is neces-
sary that only administrators have write permissions in the catalogue. All the other
users are not in direct contact with the catalogue and need only read permission.

5.3 Geographical Information Systems

Geographical Information Systems (GIS) are a system of hardware and software
to facilitate the management, manipulation, analysis, modeling, representation and
display of georeferenced data to solve complex problems regarding planning and
management of resources (NCGIA\textsuperscript{3}, 1990). In this section, I introduce the GIS
management in databases. More precisely, I present the PostgreSQL PostGIS extension
that allows the storage and querying of geometrical objects.

\textsuperscript{3}National Center for Geographic Information and Analysis, http://www.ncgia.ucsb.edu/
In the UNOSAT case, each picture provides the geographical bounding box as part of the metadata. The problem of finding the images that contain a geographical point or geographical region is equivalent to find whether a point or a small rectangle are contained inside a bigger rectangle (that corresponds to the bounding box).

PostGIS geometrical extensions to PostgreSQL are based in the OpenGIS standards of OGC\(^4\) (Open Geospatial Consortium). It provides a set of functions that allow users to store geometrical objects (Point, Line, Polygon, etc.) in the database. Queries such as which polygons contain the \((x,y)\) point can be made over these objects.

The most important functions to create geometrical objects, like points, lines and polygons from strings (GeomFromString, PolyFromString, etc.), store them, and make queries about them (Contains(Obj1, Obj2) which returns true if Obj1 contains Obj2 or false otherwise) have been added to AMGA.

### 5.4 Benchmarks of Index Usage for PostGIS

Perhaps the most important feature of any database is the possibility to create and use efficient indexing techniques. Indices allow better performances in queries over a column by organising the data in a search tree that can be quickly traversed. In traditional relational systems, B+-trees [27] are often used for the scalar types.

With more complex variables, like space coordinates \((x,y)\) the simple B-Tree approach cannot be used, and in order to create effective indices, other structures are needed. In PostreSQL (and PostGIS) the Generalized Search Trees (GiST) [28] are used for indexing. GiST are, in fact, abstract objects that possess a defined structure (balanced tree, variable fanout), key methods (user-defined methods for the keys) and tree methods. By defining properly these abstract parts, the GiST can act like other well-known search trees like R-trees and RD-trees.

The setup of the test presented in this section consists of a set \(P\) of 100 random points. The domain were we define the polygons is \([0, l] \times [0, l]\), with \(l = 10,000\). We insert \(N\) random square polygons with edge length \(\frac{l}{\sqrt{N}}\) in the AMGA directory "polyN" which contains an attribute \texttt{id} which is an integer and another attribute \texttt{poly} which is a Geometry. \(N\) ranges between \(10^3\) and \(10^6\). We measure the time that it takes to find out the polygons that contain the points in \(P\). This operation is repeated with and without an spatial index. The log-log plot of the results can be found in the Fig. 5.1.

The reason for the choice of the edge length of the polygons according to the number \(N\) allows us to reduce the overlapping polygons (the output retrieval would influence the query time), while trying to cover all the available space (this is harder to achieve since polygons are random).

Using an index improves considerably the query time. For \(10^6\) polygons on the table, which is a reasonable number of entries for a database, the query without

\(^4\)http://www.opengeospatial.org/
Figure 5.1: The plot shows the measurement of the time needed to perform 100 queries (point belonging to a set of polygons) to AMGA. With relatively large tables, the speed gain ranges from a factor 3 to almost 100.

The time for a sequential scan, done when no index is used, should be linear in time (all the table must be traversed). However, when the number of entries is of the order of $10^6$, the table is too big to be put completely in memory forcing to reload the table from the hard disk, which is even more time consuming.

This test shows that the overhead due to the usage of AMGA (translation of the AMGA queries to PostgreSQL dialect) is not important compared to the time taken by the queries and remains almost unnoticeable.

5.5 Metadata Catalogue Schema

In this section I present the database schema for the satellite image metadata I developed. As all the image metadata will be translated to the standard set of
attributes defined by Geonetwork, the schema must have at least all the mandatory fields. In order to keep track of the images on the grid, additional fields indicating the LFN and GUID of the image must be added. The schema takes into account the possibility to have raw and processed colour images as well.

In the Fig. 5.2, I show the proposed schema for the metadata. Each rectangle (a table) corresponds to a directory with the correspondent attributes in AMGA. The lines show a foreign key dependence between the directories.

The directory **PictureInfo** is the central table in the schema. It contains all the mandatory fields presented in the Geonetwork section above. In addition, it has an id, that is the primary key of the directory and the LFN and GUID of the image that are unique. This two attributes allow to get the physical location of an image inside the SE.

The directory **Contact** contains all the necessary information about a contact that might be the person that filled the Geonetwork metadata or the point of contact for the image. Each entry in this directory possesses an integer id, that is the primary key of the directory.
The dates and their type are contained inside the Date directory. They are related to the PictureInfo directory by the idImage. Each image can have different dates, for example processing, creation, update, etc.

The Keyword directory contains a keyword type and a value. For example, an image can have a keyword of type place that indicates the country, and another keyword of type satellite which tells which satellite took the image.

The Dimension directory describes the size and the name of each dimension of the image. These fields are important to map a longitude and latitude to a point in the image (this requires also the denominator of the image).

Finally, if a set of raw images is inserted, an entry must be added to the RawDataInfo directory, indicating the unique id, the name and the source of the raw data set. Then for each of the images that belong to the set we add an entry in the RelationKind directory, that relates the id of the raw set to the id of the image. The relationKind field takes the value “bandOf”. If we insert an image that has been obtained from a raw image, we insert the necessary ids, and the relation kind is set to “generatedFrom”.

I have provided several Python scripts to work with this catalogue. One script to generate the hierarchy of directories with the corresponding fields and types as defined above. Another script to parse a Geonetwork XML file and extract the needed information from it, to insert into the metadata catalogue. This last script requires the LFN and GUID to insert the complete metadata.

5.6 Discussion

The definition of the satellite image metadata schema is an important part of the UNOSAT grid project, because it contains all the relevant information about images. The rest of the parts presented in the architecture (chapter 4) rely on this catalogue to find images satisfying the user’s criteria.

The Geonetwork web service is able to manage image metadata in form of XML files. However, it does not have an internal schema to store the metadata. Instead, the XML files are inserted directly as binary objects in Geonetwork’s internal database.

To show that this approach is not suitable for UNOSAT I give a simple example. We need to find the images that contain a point of given coordinates using Geonetwork. To solve the problem, all the XML files in the database must be parsed to extract the bounding box and test if the point is inside it or not (this test should be implemented by hand). If the number of XML files in Geonetwork is small (some hundred images), this approach works in an acceptable time. Nevertheless, as the number of XML files grows, such a query involves a lot of computations that can take a lot of time. In particular, the performances of such a file-based system cannot match those of a similar system using PostGIS, that stores the bounding box as a Geometry and has an index over this column.
The keyword system used in Geonetwork is very interesting. It uses a database table dedicated to store such information for each image, and finding images that correspond to a certain keyword is easily done by querying this table. The keyword directory is implemented in the metadata catalogue using the same ideas.
Chapter 6

Web Service Prototype

In this thesis I used several prototypes to progress and validate the different ideas and to test different technologies. Prototypes are quickly-implemented incomplete models that allow to test a set of the features of future full-featured software programs being developed. In addition, also the final users can be exposed to the system to detect misunderstanding on the requirements or potential performance problems. To test and validate the usage of the satellite image metadata catalogue created with AMGA, a web service prototype has been developed. The prototype implements a simple authentication method based on grid proxy certificates, a query interface that allows the user to choose the pictures that best suit his needs and the possibility to download the pictures that interest him directly from an SE.

In the following, I describe the development and main features of this prototype service. First of all, the workflow is explained, in order to show the basic operations that the prototype must implement. Then, I present the implementation details of the prototype, starting with a brief introduction to the chosen technologies for the prototype and explaining the usage of the prototype. I also have developed a lightweight prototype for a mobile phone, based on the same web services, in order to show the usage of the proposed technologies from a simple mobile device. This mobile phone prototype is presented in the next chapter.

6.1 Workflow

The workflow of the web service prototype consists of three main parts: upload of grid proxy certificate, query of the metadata catalogue and download of images. A limitation of the prototype is that no image processing is implemented, as it was required in chapter 4. The workflow of the prototype is shown in the Fig. 6.1.

The user starts providing a valid grid proxy certificate to the server. A service verifies that the proxy is still valid (i.e. it has not expired), that the SE can be accessed with this proxy and that the certificate is accepted by the AMGA server.

Once the user has correctly provided his credentials, he receives a cookie (with
Figure 6.1: Workflow of the usage of the web prototype. The user sends his proxy certificate, and if it is valid he receives a cookie as proof that he has been authenticated. Then he queries the metadata catalogue. He gets a list of the images that correspond to his selection criteria. He can download the images that interest him afterwards.

same validity time as the proxy certificate) to identify him to the service in future connections. He becomes an authenticated user and can start using the prototype query interface. First of all, the user queries the metadata catalogue for images following certain criteria by using some of the implemented metadata selectors. Given the variety of the metadata a big number of selection criteria can be implemented, such as selection by country or by keyword. The query returns the names of the images that correspond to the user’s criteria. Then the user can ask further information about each of the images returned in the above step. This gives more flexibility to choose the images and refine the query. Finally, when the user has found the image that interests him, he can proceed to download it, via the web service, from the SE. The process can be repeated as many times as it is necessary and while the user remains authenticated (his cookie is still valid).
Figure 6.2: Web page of the prototype. The selectors are found left. After having used the selectors, the titles of the images that correspond to the criteria are shown in the middle part. Clicking a title of an image retrieves more metadata on this image along with the possibility to download it.

For the sake of simplicity, there are no compression or processing steps on the prototype, given that its goal is to test mainly the usage of the metadata catalogue. Therefore, to be compliant with the proposed UNOSAT grid application architecture (chapter 4) this step should be considered and implemented.

6.2 Implementation

In order to allow a rapid implementation of the prototype it has been designed as a web service (Fig. 6.2). The primary prerequisite is that the prototype service runs on a machine that is a grid User Interface (UI) to be able to use the LCG command line tools. The choice was to use a server running Scientific Linux Cern (SLC) with an AFS installation, because a complete UI installation is provided on AFS at CERN.

Besides from Hypertext Mark Language (HTML), other Web technologies have been used to make the site more practical and easier to use, notably Javascript, Asynchronous Javascript + XML (AJAX) and Common Interface Gateway (CGI) that are executed from an Apache web server. In the following, I will give a brief overview of each of these technologies explaining why they were chosen.
Javascript is a dynamic scripting language for Web programming, developed by Netscape, that can be used either as an object-oriented or as a procedural language. It is widely used to add dynamic effects to Web pages. The syntax is very similar to C++ or Java, so the language constructs work in the same way as the ones in these languages. Javascript is a superset of the ECMA-262 Edition 3 (ECMAScript) standard scripting language.

AJAX is a name, coined by J. J. Garret in 2005 [30], for a technology that allows asynchronous queries from the client to the server. The normal flow of a Web request is that the client sends a query to the server, waits the response and reloads the entire page when it arrives. The AJAX approach is to send the query asynchronously, letting the client do other things while the query is processed on the server, and then dynamically update the page as the answer arrives. A Web application using AJAX can therefore achieve the responsiveness of a desktop application, of course given a certain quality of the network connection. The key to achieve this is the ActiveX Microsoft.XMLHTTP object for Internet Explorer and the XMLHttpRequestObject for Netscape based browsers. These objects can send asynchronous queries and can retrieve the response as XML or text as soon as they are accomplished.

Javascript and AJAX technologies were used on the implementation of the UNOSAT web service to provide more ease of use while querying the metadata catalogue.

The Common Gateway Interface (CGI) allow us to run external programs on the server side. The Apache web server supports CGI scripts in several scripting languages. CGI scripts allow to use the AMGA Python API to query the AMGA server and shellscripts to use the LCG command line tools necessary to verify a proxy certificate and to download an image from an SE to the server.

In the next sections each of the different parts of the implementation are explained.

Authentication

In the authentication part, the user is asked to provide a valid grid proxy certificate to the Web server. This proxy certificate is sent to the server via an HTML file input. The grid proxy must be created by the user previously, by logging into a UI that contain his certificates for example. This method is far from being optimal, because the user needs to deal with proxy certificate generation, and must therefore have some knowledge on grid authentication.

The test server uses a CGI script to write the proxy certificate sent by the user, to test if it is still valid and if the user can access the AMGA catalogue with it. The grid proxy certificate is needed on the server to send queries to the AMGA metadata catalogue and to download the desired images from the UNOSAT dedicated SE.

If the proxy certificate of the user is valid (i.e. it has not expired, it can access the SE and AMGA), then the server will send a limited duration cookie to the user (same duration as the proxy certificate). This cookie contains the location of the proxy certificate of the user on the server and while it is valid the user can access
the web service query service without providing his proxy certificate. When the cookie expires, the user is redirected to the login page where he is requested for the proxy certificate again. It must be noted that the cookie itself is not used for authentication, it serves only to access the users proxy certificate, but not possessing a cookie means that the user proxy certificate cannot be found on the server.

This simple authentication method was chosen to allow a faster implementation of the prototype by skipping the delegation methods. The security model of the EGEE grid would however require to delegate a restricted proxy certificate from the user’s browser to the web service. In practice this is not feasible, because the web browsers lack the capability to delegate proxy certificates. An alternative way of authentication is done by the UNOSAT application presented later, where the users have accounts on the server machine. A password is used to log on to the server and to unlock the proxy certificate.

**Query Interface and Download of an Image**

The query interface is accessible once the user has been authenticated. The user can choose between several selectors which offer different categories to query the catalogue. The web page proposes currently a graphical selector (Fig. 6.3, where the user can select the region that interests him by clicking and zooming in to a world map. This maps are created on the server side with GMT (Generic Mapping Tools [31]) by using a CGI script, and then sent asynchronously to the client by using AJAX technology. The user can choose to put some coordinates by hand as well. Other selectors allow to choose either processed, raw images (the user might want to process them himself) or both. Given the variety of metadata that each image possesses, many other selectors such as keyword selector, image format selector and others could be implemented in order to make queries more precise.

After the user has chosen the values on the selector that he wants, he sends a first asynchronous query to the catalogue. A CGI script is then executed to communicate with AMGA and to fetch all the images IDs that correspond to the user’s criteria. This information is returned as an XML file to the client. With the help of Javascript this XML is parsed and the pertinent information is extracted from it. Then, the updated information is added to the page dynamically. The user can get more information on each of the files that have been returned by this query. This way, the user can choose precisely what images he is interested on. Again the retrieval of metadata on each

Finally once the user has chosen an image that interests him, he can download it by clicking on a hyperlink over the logical file name of the image which was retrieved from AMGA. This click calls a CGI script that uses a shell script to call the `lcg-cp` [9] command that copies a file from a grid SE using its LFN (chapter 2). The server uses the proxy certificate to copy the image from the SE and then sends it to the client.

Actually, as images can be quite big and no processing or compressing is applied
Figure 6.3: Graphical selector to choose precise regions of the planet in an easier way. On the left, the initial situation presented in the prototype. On the right, the result after having clicked over Europe. These images are generated in the server, as the client requests them, using GMT.

to them, this step might require the client to wait for around 10 to 15 minutes for a 500 Mbytes image.

6.3 Discussion of the Prototype

The prototype allows us to test the usage of the satellite image metadata catalogue. There are plenty of selectors that can be implemented, this because of the several fields in the metadata catalogue. The prototype requires the browser accessing the service to include Javascript support.

For a final service, it is necessary to add the missing parts, that are, authentication and processing of the images with grid jobs. The authentication, as discussed in the previous section, should be implemented with proxy delegation. For this, it is necessary that delegation methods for web browsers are implemented by the people working with security.

A job that processes the image would improve the performances of the prototype, because the server would not download the whole image, but only a compressed part. However, depending on how busy the CE are, the time taken by a job can also be increased.

Usage of technologies like AJAX or CGI scripts in the prototype proves to be a good choice, since they simplify the operations on the web page to the user. It would be very interesting to have such technologies in a final application.
Chapter 7

Mobile Phone Prototype

As part of the requirements of UNOSAT for the grid project (chapter 4), users need to access it from a mobile device such as a mobile phone. The web service prototype presented in the last chapter requires that the web browser used to access it supports Javascript. Even though modern mobile phones implement web browsers that can access web pages using the WAP protocol, not all of them have Javascript support. In addition the very limited bandwidth and long latencies make conventional web pages very difficult to use on a mobile phone. Therefore, it is necessary to have another prototype based on the existing components, that can be deployed and used in mobile devices, which offers similar services as the web service prototype.

In this chapter I describe the development of a mobile phone prototype, written in The Java 2 Micro Edition (J2ME). The J2ME platform is widely used to develop applications for mobile devices. This prototype allows users to download image from the grid based on its metadata. As it is impossible to work with certificate authentication or even create proxy certificates in a mobile device due to API restrictions, the mobile phone prototype does not implement any authentication scheme. In the future, however, it is planned to do authentication based on the phone’s unique telephone number.

In the following, I introduce the Java 2ME technologies that were used to develop the prototype. Then I explain the web services that the phone application contacts in order to retrieve satellite image metadata from the catalogue, to be able to access and download smaller versions of images stored in the grid SE.

7.1 Java 2 Micro Edition

J2ME is a set of technologies and specifications, that adapts the existing Java technologies and creates new ones, in order to work with mobile devices. The syntax is the same as Java 2 Standard Edition. Not all standard Java classes are included in J2ME because of the limitations of the devices supports. Some other classes have been rewritten, in order to be compliant with mobile devices.
A Connected Limited Device Configuration (CLDC) [33] defines a Java virtual machine for limited devices such as mobile phones and low-end PDAs. The Mobile Information Device Profile (MIDP) [34] defines the execution of applications and management of user interfaces, persistent data, access to external servers and secure access to data for CLDC. MIDP is a layer above CLDC, it provides all the high-level functionalities. MIDP 2.0 [35] is the latest revision of the MIDP architecture.

A MIDP application is called a MIDlet. MIDlets are distributed in a .jar file, together with an associated Java application description (.jad) file. The Java Wireless Toolkit\(^1\) makes the development of MIDlets easier, because it allows testing and debugging the MIDlets in a personal computer, before deploying them on a actual mobile device.

### 7.2 Implementation

The implementation of the prototype consists of a simple MIDlet, written according to the MIDP 1.0 specifications to allow compatibility with a broad range of mobile phones providing a J2ME virtual machine, and a server that provides services contacted by the MIDlet. In the following I present the intended workflow of the phone MIDlet. Then I explain the implementation details and technologies used for the mobile phone prototype.

An example workflow of the prototype is shown in Fig. 7.1. The user provides the desired coordinates through a simple interface. These coordinates are sent to the server, which queries the metadata catalogue in order to get the images that contain the coordinates. The list of such images are sent to the user in form of a list. The user can select any of the images in the list. This operation makes the server download the selected image from an SE. Satellite images are too big to be sent back to a mobile phone. Therefore, they must be cropped around the coordinates that the user provided to get a small rectangle (150 by 150 pixels) that can be correctly displayed in a mobile phone. The cropped image must also be converted to PNG format. The URL of this processed image is sent back to the user’s phone that downloads it and displays it.

The mobile prototype contacts CGI scripts executed in the server that provide similar services to those used by the web prototype presented in the last chapter. The only difference consists in the encoding of the answer messages sent by the server to the mobile client. XML is not used anymore, because of the relatively large memory consumption during the parsing, which makes XML problematic to use in small devices. Furthermore, data transmission is billed by the volume of the exchanged messages. Therefore, messages are sent as simple encoded strings.

After sending the coordinates to the service, a CGI script contacts the AMGA server to find out which pictures contain the given coordinates. The server sends

\(^1\)http://java.sun.com/products/sjwtoolkit/
Figure 7.1: Example usage of the mobile prototype within the J2ME wireless toolkit simulator. The user inserts the coordinates (left), then he gets the titles of the images that covering them (center). The user can choose any of these images to get a cropped version of the image centered around the coordinates (right).

back the title(s) of the image(s) and the LFN(s) of the file(s) in a single string, encoded as follows:

title1\LFN1\title2\LFN2\...

The phone application retrieves the information from this string as it is read. Then, it uses this information to create a list menu with the titles. When one object of this list is selected, another CGI script is contacted on the server. This script takes the LFN of the file that interests the user, retrieves it from the SE, crops it and converts it to PNG format. PNG is the image format supported by most of the mobile phones in the market. The CGI script to copy files from an SE is almost the same as the one used by the web prototype, the only difference being that the CGI script contacted by the mobile uses a previously stored proxy certificate (because there was no authentication).

7.3 Discussion

The choice of developing a mobile application instead of using a web browser on the mobile phone to contact a web page was made to avoid dealing with all the different browser particularities in mobile phones. Communication via short string messages with CGI services running on a server is faster and requires less information exchange between the entities.
For the sake of simplicity the prototype did not implement any authentication. In order to provide a complete mobile phone application, the authentication steps must be added. As certificates cannot yet be used from mobile devices, grid authentication is hard to implement on them. One way to go around the problem would be to implement a login/password authentication on the mobile phone that maps to a grid authentication. Another possibility would be to use mobile phone’s unique telephone number. For example the service could sent an SMS to the phone with a shared secret. SMSs can be easily accessed from J2ME’s API to retrieve the secret and use it to secure the communications with the server.

Another interesting improvement that can be made to the prototype is to add an interface to either internal GPS devices or external GPS devices via Bluetooth. While this provides a large improvement in the ease-of-use, the portability of such a solution would be limited because of proprietary GPS interfaces.

The prototype was successfully installed in a Siemens S65 mobile phone via Bluetooth. The usage was shown as part of the UNOSAT grid project demonstration in the 2006 EGEE Conference held in Geneva (September).
Chapter 8

UNOSAT Web Portal

The UNOSAT Web Portal (Fig. 8.1) is the result of a collaboration between CERN, UNOSAT and the Italian company NICE srl. The web portal is used to upload images with their metadata and to deliver processed images to users. It can be accessed from mobile devices or a web browser. It is based on the UNOSAT grid architecture presented in the chapter 4. This web portal was part of a UNOSAT demonstration in the EGEE conference held in Geneva in September 2006.

The development of the portal involved many people that contributed different technologies to get the final result: CERN provided the metadata catalogue schema and several AMGA scripts to query and to populate it, that have been developed in this thesis. The UNOSAT project provided the images with their Geonetwork-based XML metadata, as well as a job description to process the image on the grid based on the metadata of the image. NICE srl. provided the web portal technology, based in GENIUS and Enginframe framework. This portal must present all the applications developed previously in an easy-to-use interface for the user. Finally, WLCG/EGEE provided the grid infrastructure used from the portal, namely dedicated RB, CE, LFC, SE and a VOMS for the UNOSAT VO.

In this chapter I detail the conception of this web portal. I start by introducing the GENIUS portal technology. Then, I explain the implementation steps that were made to achieve the final product. The last part discusses the choices made and proposes future work on the portal.

8.1 GENIUS Portal and Enginframe

The GENIUS (Grid Enabled web eNvironment for site Independent User job Submission) [36] portal offers the users the possibility to interact with the grid in a graphical way through using a web interface. The motivation for such a portal technology is to overcome the usage of the CLI offered by the gLite middleware that discourages the novice users to use the grid technologies. GENIUS is a project started by the INFN and NICE srl. that provided the Enginframe XML framework.
The GENIUS portal works on top of the grid middleware, alongside with an Enginframe framework and an apache server. All these elements are installed into a machine containing also the UI infrastructure (in order to have access to the grid). On the client side, the use only needs a web browser to access the portal.

GENIUS allows users to authenticate and create grid proxy certificates either by storing their certificates into an account on the UI or by providing the .p12 from of the X.509 certificate. In both cases the password that unlocks the proxy certificate must be provided, but as secure http connections are used to contact the portal, this operation is secure. Other capabilities of the portal include job submission and access to data stored on the grid.

Several communities have tested and currently use the GENIUS portal technology, such as the ALICE and ATLAS experiments at CERN. GENIUS is used in EGEE as a training tool: new applications or user community start to use the grid via the portal as a preliminary step before using the production infrastructure. Some application are actually using the grid only via portals.

### 8.2 Implementation

The functionalities of the portal have been implemented as command line applications, because the Enginframe technology allows the insertion and execution of such
applications from a web page.

The following services are made available to the: authentication, metadata insertion and uploading of a single image to an SE, metadata insertion and uploading of Landsat 7 raw bands and job submission to download a processed image by providing the coordinates.

Every user must have a traditional Unix account on the portal server. The login and password for this account are used to access the portal. Once the user has logged in to the portal with the credentials for this account, he can start using the features of the portal. The first step he needs to take is to create a grid proxy certificate. The portal requires the .p12 form of the X.509 certificate installed in the web browser and the password to create the proxy certificate. As all the communications are secured, the information can be transmitted with no risks. Once the proxy certificate has been created, the user can ask information about it and destroy it when login out.

The metadata insertion and image uploading is implemented by two scripts, one that populates the metadata catalogue using the Python AMGA API, written by myself, and another that makes the copy and register of the image to the grid written in shell script. When inserting raw images, the previous scripts are used in loops to insert all the files. After running all the population and copy and register, additional information about the raw image set is inserted into the catalogue.

Job submission is the main feature of the portal. The user inserts the coordinates and a radius in the web page (Fig. 8.2). This radius defines the length of the area around the coordinates that the user needs around the coordinates, measured in meters. The portal runs an application that queries the AMGA satellite image metadata catalogue for images that contain the coordinates provided by the user. If one image is found, then it is chosen. On the other hand if several such images are found, the latest one is chosen by default. The image coordinates that correspond to the radius are computed by this application. A job description is created including the LFN of the image (converted to the SURL by the LFC in the job) and the coordinates that correspond to the region of interest of the user (defined by the aforementioned radius). The job description is submitted to the RB that will select a CE for the job after the matchmaking.

The FWTools\(^1\) package must be installed first on the selected CE to execute the job. The FWTools package provides a framework to work with GIS, including tools to process images (cropping, converting formats, etc.) and has been stored in a SE from where it is copied to the CE by the job. The selected image is also downloaded from the SE to the CE. It is cropped according to the computed coordinates and it is compressed in ECW and JPEG formats. Finally, this processed images are downloaded from the RB to the UI machine and from there, they are sent back to the user (Fig. 8.4).

The status of the job is available for the user, at all times, and it is updated automatically (Fig. 8.3). The complete job submission and execution takes from 10

\(^1\)http://fwtools.maptools.org/
The portal has an interface to take the user coordinates and a radius. to 15 minutes while making a request of an image of 500 Mbytes.

The portal cannot currently work from a mobile phone, because the web page requires a Javascript compliant browser, which is not always the case in today’s phones. This issue will be addressed in the continuation of the project.

8.3 Discussion

Although the UNOSAT portal is still under development, it is already very usable and shows all the basic functionality that was required. The job submission and monitoring features, which were the main focus in the development of the portal, are very elaborate and the portal hides the grid infrastructure and authentication mechanisms very well from the user.

All the services (CLI applications) were implemented inside the portal in close collaboration with the NICE team. The services and ideas that I provided for the two prototypes developed before the portal were included in the portal.

A variety of programming languages were used in the pieces of the project because of the number of people involved in the project. The insertion of metadata in the catalogue is implemented in Python, the copy and registration of an image into the grid uses a shellscript, the queries to the metadata catalogue are written in Java and the job submission is implemented, again, in shellscripts.

Another problem that need to be revisited in the portal, in order to provide better services, is the mobile phone compatibility problem, that should be solved in a different way, creating a dedicated Java ME application that contacts services running in the portal, instead of accessing a web page.

In the future, more features must be added to the portal like more flexibility to the user on the choice of the images that he needs and submission of jobs with more complicated processing of the images. For this, my studies on the user interface to the metadata from the prototypes can prove to be very useful.
Figure 8.3: The user gets the status and information of all jobs that have been submitted to the portal. A user can always choose to cancel a job that has been submitted or is already running.

Figure 8.4: The processed image arrives to the user after the job has finished running. The user receives a JPEG image (to show on the browser) and a ECW image.
Chapter 9

Conclusions

New communities are joining the grid for the benefits of shared computing, storage and other resources. This is also the case for the UNOSAT project, which needs large amounts of storage space for the satellite images they possess and has large varying demands of computing power to process them. The UNOSAT grid project aims to develop an initial grid service for UNOSAT to deliver processed images to field workers using mobile devices or a web browser. The UNOSAT web portal is the practical implementation of this grid service.

Satellite image metadata provides important details about the image it describes. In order to have efficient access to this metadata, a dedicated metadata database schema must be conceived. In this thesis, I have presented a satellite image metadata schema that has been tested and used in two prototypes and the UNOSAT web portal. The choices of design provide the required functionalities to the applications that use it.

To conceive this metadata schema, I have studied several different formats of satellite image metadata. However, as UNOSAT adopted the Geonetwork standard for their metadata, the natural choice was to base the schema on this standard. Since Geonetwork only proposes an XML schema, it was necessary to organise the information contained in the XML files into a complete database schema, defining primary keys for each table and foreign keys to establish the relationships among tables, while ensuring good performances and easiness of access to the information. Grid related information (LFN, GUID) had to be also taken into account, as well as other use cases (raw images and processed images), enhancing the schema further.

The catalogue implementing the schema was created with AMGA. To query the catalogue from the prototypes and the UNOSAT web portal, the different APIs provided by AMGA (Python, Java, C++, etc.) have proved to be extremely useful, as they allow the developer to choose the programming language to be used. AMGA also allowed the implementation of certificate-based authentication for the web service prototype.

The development of the prototypes to test the satellite image metadata catalogue allowed the discovery of interesting technologies in web services (AJAX) and for
mobile phone programming (J2ME). The prototypes have given valuable feedback about the utilisation of the metadata catalogue.

The UNOSAT web portal is an ongoing effort. However, it is an innovative application for the grid, that will be soon deployed to be used in real situations by people that need to have access to high-quality satellite imagery from remote places in the world.

In the future, the portal must offer more services like browsing the information of the images that have been uploaded and allowing more flexibility in the choice of images to the user. Further and more complicated processing of the images, which are time consuming, should also be implemented in the portal, to explode the computing power that the grid offers. One of such processing is combining raw spectral bands to create a color image.

The possibility to access the web portal from a simple mobile device must be a priority for the continuation of the portal development so that it can be tested by real users on the field. These users will surely provide valuable feedback about the utilisation of the portal. A choice must be made between accessing directly the portal with a web browser or installing an application that contacts services accessing the grid, on the phone. The first option depends on the browser of the mobile device (Javascript compliant or not) and needs more information to be exchanged. An application for the mobile phone takes more time to be implemented, but data exchanges are lighter. In both cases the certificate-based authentication is not possible to implement, given that this kind of authentication is not available for mobile devices.

Hopefully, the portal will provide valuable assistance to the field workers in the future when the occasion requires it.
Acknowledgments

I would like to thank my supervisor from the University of Geneva, Prof. Bastien Chopard, for giving me the opportunity to work at CERN.

I would also like to thank Dr. Massimo Lamanna for allowing me to be part of his group at CERN, which was an enriching experience for me.

Finally I want to thank the people from the ARDA group with whom I have worked at CERN, specially Dr. Birger Koblitz. His valuable advices helped me understanding and solving the problems faced during the realisation of my master thesis.
Bibliography


