RINGrid contract no. 031891

Evaluation and Requirements for Infrastructures

Deliverable 3.2

Status of Grid Middleware and Corresponding Emerging Standards for Potential Usage in Sharing Scientific Instruments via (International) Networks

Document Filename:	RINGRID-WP3-D3_2-JKU-Middleware.odt
Work package:	WP3
Partner(s):	JKU, PSNC, GRNET, TUI, REUNA, RNP, CNIT, CNR/ISTI, UNIS
Lead Partner:	JKU
Document classification:	Public

Abstract

Grid computing is a broadly accepted paradigm for sharing resources like processing power, or storage space. However, the grid is based on such a broad concept that it can do more than just integrating computing facilties. For example, it can be used to access expensive and thus not so easily available instruments. However, the middleware for instrument integration is not as well-polished as the middleware which is available for integrating computing facilities.

This document gives an overview of the the solutions which are available, or are currently emerging, for remote instrumentation. Since remote instrumentation is a topic currently heavily under research, there are not many solutions available. Thus, this document concentrates on the solutions which are available, such as CIMA, as well as competing projects, such as GridCC. It also gives some use cases which help identify weak spots in the currently available technologies.

Delivery Slip

	Name	Partner	Date	Signature
From	Thomas Prokosch	GUP		
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Authors
1	22-Mar-2007	Initial draft	Thomas Prokosch
2	26-Mar-2007	Extended draft based on the findings of the teleconference held on 22-Mar-2007	Thomas Prokosch, Constantinos (Costas) Kotsokalis, Michael Stanton, Franco Davoli, Antonio-Blasco Bonito, Alberto Gotta, Damian Kaliszan, Tomasz Rajtar, Zhili Sun, Jin Wu
3	25-Apr-2007	Almost complete version.	Thomas Prokosch
4	19-Jun-2007	Final version incorporating the changes proposed in the review which took place at the end of April.	Davide Adami, Antonio-Blasco Bonito, Marius Branzila, Luca Caviglione, Romeo Ciobanu, Davide Dardari, Franco Davoli, Marcio Faerman, Alberto Gotta, Damian Kaliszan, Constantinos (Costas) Kotsokalis, Angelos Lenis, Fabio Marchesi, Thomas Prokosch, Michael Stanton, Jin Wu, Tasos Zafeiropoulos, Yanbo Zhou

Table of Contents

Illustration Index

1. Introduction

1.1. Definition of Middleware

Middleware provides an abstraction layer which encapsulates common services and tasks in a standard interface. When the implementation of a service changes, the software stack utilizing the service does not need to change because of the standardized interface. The middleware hides unimportant differences of software and hardware while preserving its core characteristics. This definition of middleware is the same whether one takes traditional grid computing into account, or remote instrumentation over the grid — a relatively new topic — is considered. However, for instrument middleware further considerations are necessary, which are described in detail in the next section.

Middleware is a term coined by d'Agapeyeff in 1968. Back then it was not part of the grid infrastructure but was seen as a general concept for allowing application programs to use operating system routines (see figure 1.1.1). Therefore, in a grid context middleware is defined as those parts of the grid infrastructure which lie between the operating system and the individual grid applications. In a more general definition, middleware seeks to connect not only an application to the operating system, but two or more applications together so that they can exchange data.

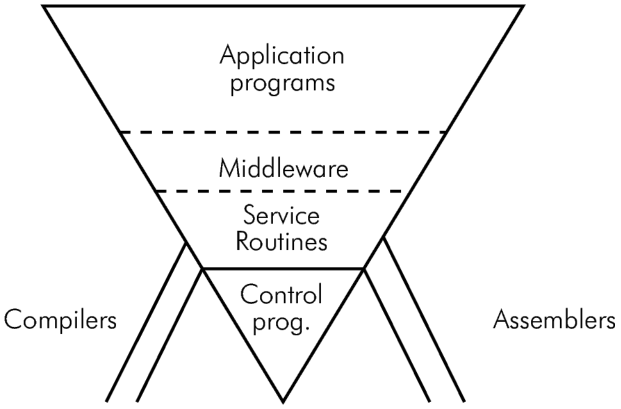


Figure 1.1.1: d'Agapeyeff's inverted pyramid.

In the grid context, middleware helps with the organization of distributed resources. Therefore, it has specifically the following tasks:

- perform scheduling and authentication
- make computational resources consistent and dependable
- provide network services discovery and binding (using remote procedures calls)
- provide access to data resources (specifically, see the DAI chapters in this document)

1.1.1. References

Naur, Randell (eds.), "Software Engineering",Report on a conference sponsored by the NATO science committee, Garmisch, Germany, October 1968.

1.2. What Differentiates Instrument Middleware from Other Middleware?

Scientific instruments (as well as sensor networks) are absolutely necessary for scientific advancement. There are some devices, which are very expensive and therefore only one or just a few pieces of this device exist worldwide. This is for example true for big radio astronomy telescopes. As a consequence, scientists who want to perform experiments on these devices have to take major exertions in order to travel to the instrument in question, if they are able to afford the journey, and they are welcome in the foreign country where the instrument is located. These are obstacles not everybody can overcome. One possible solution we are looking at is to integrate these devices into the grid infrastructure, so these devices can be operated remotely. This is where special instrument middleware comes in.

Why is instrument middleware so special? Data from instruments are seldomly used in a raw form but often have to be corrected, normalized, calibrated or annotated with some metadata. As data which is used in the grid normally does not have to undergo these steps (most of the time it is already in a form similar to what is needed), data from instruments have to be treated in a special way — as a consequence also the instruments require special treatment, i. e. special middleware. Additionally, large instruments often produce large amounts of data, which can pose a problem for networked computers, so data processing from these instruments need to be integrated into the grid infrastructure as well. This is especially true for sensor networks, a type of instrument where a multitude of sensors are being connected together in order to gain some relevant data (cf. earthquake sensors).

As already outlined, instruments are often geographically dispersed, and they are often unique or one of only a few. This as well as the real-time nature of some instruments shape the way middleware has to be architectured: It is not economic to create a distinct middleware stack for each of the devices. Not only that it will be very difficult to access devices when the middleware is different for each one, but it would be also nearly impossible to modify or upgrade instruments. Instead, it is desirable to reuse existing software with new hardware as much as possible.

Instruments are characterized by intrinsic and extrinsic parameters. Instrinsic parameters are inherent in the instrument, they cannot be changed by better hardware, but are instead determined by the scientific use. Extrinsic characteristics are determined by hardware design and can be changed as soon there is scientific progress or more money is available to build better instruments [1].

Intrinsic Characteristics	Extrinsic Characteristics
Number of data sources. There exists a wide variety of instrumens, some are comprised of only one data source, other instruments have hundreds or even more sensors which provide data. On the one side of the spectrum lies for example an electron microscope which gives only one single picture over possibly large timespan, on the other side of the spectrum are sensor networks comprised of thousands of sensors which are used to measure earthquakes. The number of data generating sources determines how the data is handled. When designing an infrastructure for instruments comprised of many data sources one has also to think of scalability and data	**Bandwidth availability.** The availability of bandwidth increases as time passes. However, some instruments inherently have high data rates, and so different techniques have to be used to reduce the data rate if it exceeds the bandwidth available. Among those techniques are data compression, data reduction or preliminary analysis. **Processing power.** Depending on the goal set for an experiment, one either needs a supercomputer for doing the postprocessing and data analysis, or one may look at the raw data and find the results. Processing power can always be increased, and so more

Intrinsic Characteristics	**Extrinsic Characteristics**
routing.	complex experiments can be conducted.
Data rate. There are instruments which take days, weeks or even months to complete one single experiment. Other instruments have much faster data rate, such as video producing instruments or instruments like the Large Hadron Collider.	**Memory availability.** As memory density increases, instruments are equipped with more and more memory. This allows for more complex or more precise experiments.
Value of a sample/timeliness. Instruments with only one data source provide valuable data, i. e. the experiment is useless if a sample is lost. This is not the case for all experiments, for example one could still make predictions about an earthquake when one out of 1000 sensors did not deliver its data in time.	

A closer association of the experimental data and the instrument from which the data originated will yield a tighter feedback loop between the experiment, the instrument and the scientists who are in charge of the experiment. This allows for experiment steering, saving time and resources, while getting better experimental setups. It is more probable that the data collected has some value.

Scientists can interact with remote devices in a number of ways:

- **Pure data collection.** The scientist accesses the remote instrument's data without exerting control over the instrument while the experiment is conducted. All the parameters are set up in advance, then the experiment is started.
- **Maintenance control.** The scientist has some means to interact with the remote instrument, but the interaction is limited during an experiment. Such an interaction could be calibrating the instrument before the experiment starts.
- **Person-in-the-loop.** The scientist has significant control over the instrument while the experiment is running. However, the scientist cannot control the instrument directly, but all operations are supervised by a local technician who takes care that no dangerous actions are carried out. This is the kind of remote operation the RINGrid project wants to deal with.
- **Remote control.** The scientist controls the remote instrument directly, without any middleman. In this case the middleware has to take into account that there possibly exist dangerous operations, i. e. operations that can harm people who are located in the vicinity of the instrument. Such operations have to be detected and disallowed.

The kind of instrument access has to be considered when creating an instrument middleware.

To sum it up, this is what the user of remote instruments expects from remote instrumentation middleware:

1. real-time data processing, interactivity for visualization (therefore low latency and jitter)
2. collaboration tools (which in turn asks for good scalability for a large userbase)
3. enough storage space for digital libraries, and either high-volume data transmission, or the ability to understand multiple protocols within one system to adjust the data rate accordingly
4. end-to-end reliability and fault tolerance

5. scalability for single, large instruments, as well as thousands of small sensors
6. online data reduction and analysis

1.2.1. References

[1] Bramley at al, "Instruments and Sensors as Network Services: Making Instruments First Class Members of the Grid", Technical Report TR588, Indiana University Department of Computer Science, December 2003.

1.3. How is This Deliverable Organized?

Web Services (WS) and Open Grid Services Architecture (OGSA) as basic building blocks of grid middleware. No grid infrastructure can exist without considering these specifications. It is necessary to understand these basic technologies since several pieces of a remote instrumentation solution are still missing, and projects deploying the remote instrumentation idea have to implement these missing parts. (The missing parts are identified in WP4.)

Workflow management is most important if experiments are conducted remotely. The reason for that lies in the fact that data cannot be managed locally, on the scientist's workstation, but instead need to be copied from or to the remote instrument's data storage, and shared between collaborating scientists. A part of the web service specification deals with workflow management, the corresponding standard is called "Business Process Execution Language for Web Services" (BPEL4WS).

In the third chapter of this document, middleware projects and components are presented. Most important is to look at what other organizations are doing in this respect, so the "Instrument Middleware Project", the "Common Instrument Middleware Architecture" (CIMA) as well as the (more basic, thus contained in the second chapter) "Interchangeable Virtual Instrument Specification" (IVI Specification) are presented. Of special interest is the CIMA, since it is a working example on how a remote instrumentation architecture could be implemented. CIMA is a project rooted in the US with the goal to build a consistent and reusable framework for shared instrument resources.

The fourth chapter presents two projects, which have been readily deployed and are using some components for remote instrumentation. One project presented is GridCC, which has developed an Instrument Element, similar to the classic Storage or Computational Elements. The other project presented is VLab, which has developed some methodologies for workflow and interactive job manipulation.

The next two chapters give examples what requirements the user expects the middleware to fulfill, and of some projects, which could benefit of better instrument middleware. The last chapter concludes this document by pointing out what we have learned from looking at current technologies and emerging standards with regards to remote instrumentation.

2. Supporting Technologies

2.1. Web Services Architecture

According to the W3C Working Group, a web service is

> "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".

Therefore, web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. The Web Service Architecture (WSA) provides a conceptual model and a context for understanding web services and the relationships between the components ofthis model.

In particular, the architecture does not attempt to specify how web services are implemented and imposes no restriction on how web services might be combined. The WSA describes both the minimal characteristics that are common to all web services and a number of characteristics that are needed by many, but not all, web services.

The web services architecture is an interoperability architecture: it identifies those global elements of the global web services network that are required in order to ensure interoperability between web services.

2.1.1. Service-Oriented Architecture

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use of DCOM or Object Request Brokers (ORBs) based on the CORBA specification. In the grid context, the term web services refers to the technologies that allow for making connections. Services are what you connect together using web services. The combination of services — internal and external to an organization — make up a service-oriented architecture.

Services. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services.

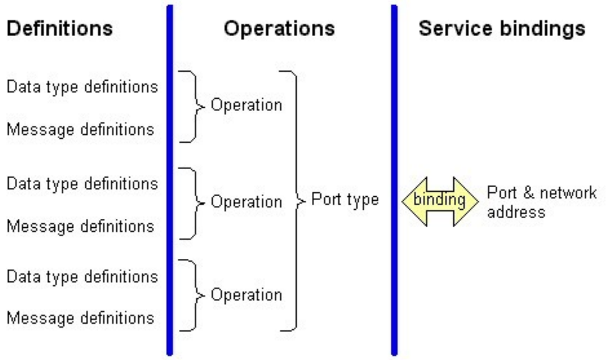


Figure 2.1.1: Definitions for web service creation.

Services are connected together using web services. A service is the endpoint of a connection. Also, a service has some type of underlying computer system that supports the connection offered. The following web service specifications have been introduced:

- **Web Services Description Language (WSDL)** is a format for describing a web services interface. It is a way to describe services and how they should be bound to specific network addresses. Figure 2.1.1 shows the relationship among the three basic parts (definitions, operations, service bindings) of WSDL.
- **Web Services Policy Framework (WS-Policy)** provides a general purpose model and corresponding syntax to describe and communicate the policies of a web service.
- **Web Services Dynamic Discovery (WS-Discovery)** defines a multicast discovery protocol to locate services.
- **Web Services Metadata Exchange (WS-MetadataExchange)** defines three request-response message pairs to retrieve three types of metadata: one retrieves the WS-Policy associated with the receiving endpoint or with a given target namespace, another retrieves either the WSDL associated with the receiving endpoint or with a given target namespace, and a third retrieves the XML Schema with a given target namespace. Together these messages allow incremental retrieval of a web service's metadata.
- **Web Service Endpoint Language (WSEL)** is an XML format for the description of non-operational characteristics of service endpoints, like quality-of-service, cost, or security properties.

Connections. The technology of web services is the most likely connection technology of service-oriented architectures.web services essentially use XML to create a robust connection.

Figure 2.1.2 illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

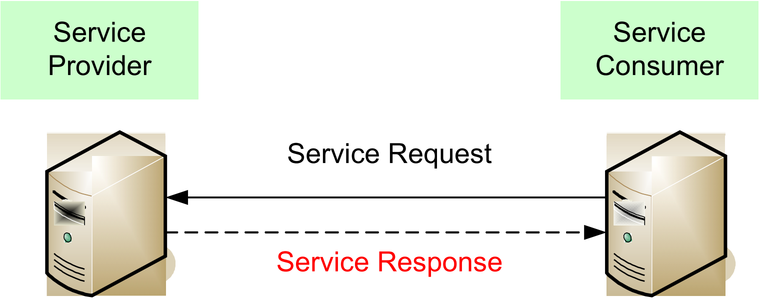


Figure 2.1.2: Service-oriented architecture.

2.1.2. Using the Web Services Description Language (WSDL)

The Web Services Description Language (WSDL) forms the basis for web services. The following figure illustrates the use of WSDL. At the left is a service provider, at the right is a service consumer. The steps involved in providing and consuming a service are:

1. A service provider describes its service using WSDL. This definition is published to a directory of services. The directory could use Universal Description, Discovery, and Integration (UDDI). Other forms of directories can also be used.

2. A service consumer issues one or more queries to the directory to locate a service and determine how to communicate with that service.
3. Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.
4. The service consumer uses the WSDL to send a request to the service provider.
5. The service provider provides the expected response to the service consumer.

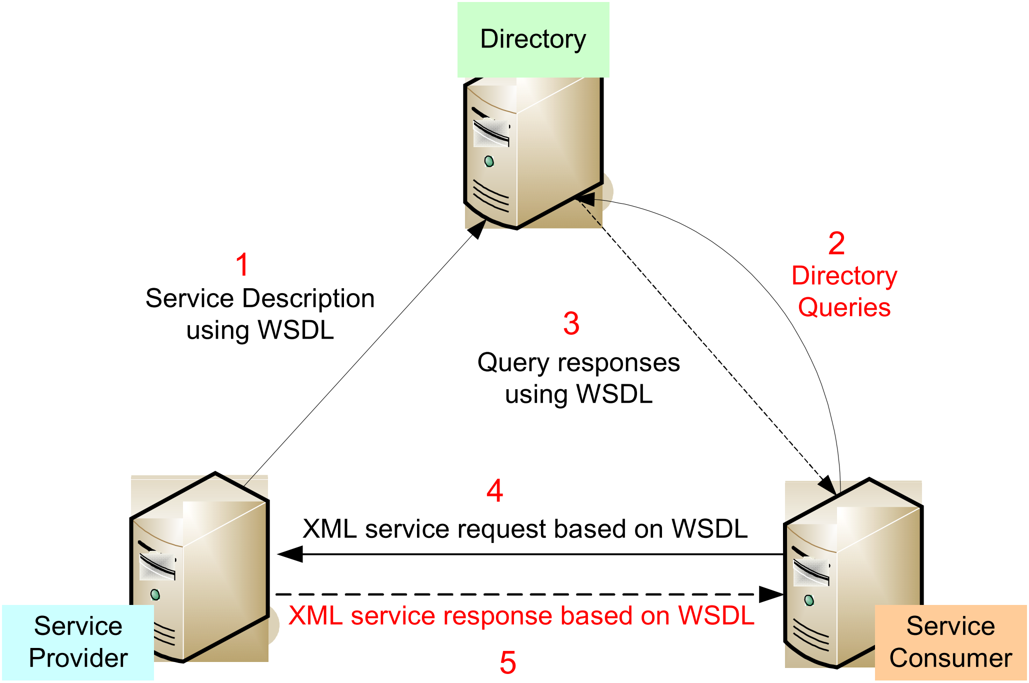


Figure 2.1.3: Consumption of a web service using WSDL.

2.1.3. Using Universal Description, Discovery, and Integration (UDDI)

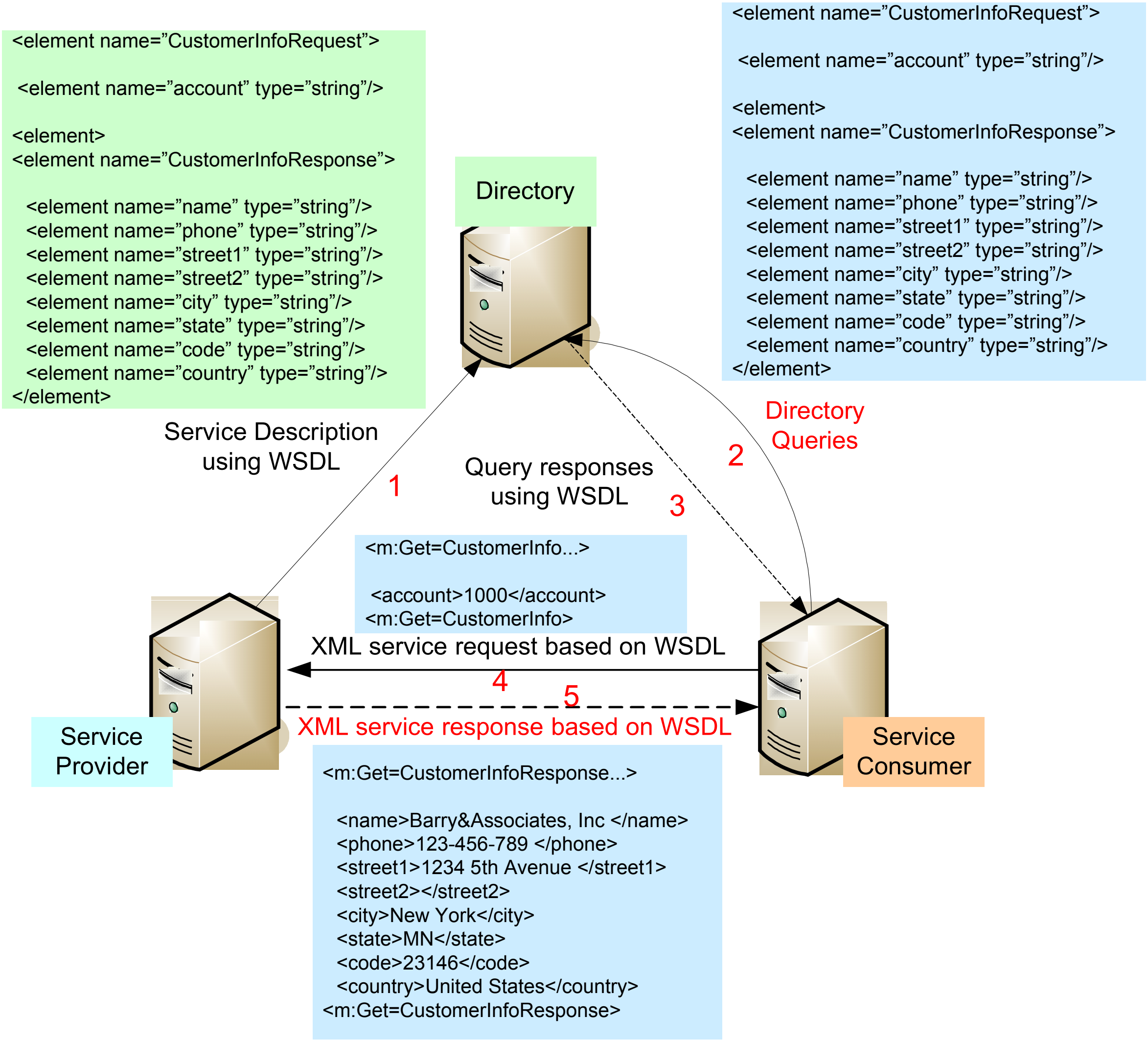
The directory shown in the above figure could be a UDDI registry. The UDDI registry is intended to eventually serve as a means of "discovering" web services described using WSDL. The idea is that the UDDI registry can be searched in various ways to obtain contact information and the web services available for various organizations. An alternative to UDDI is the ebXML Registry.

2.1.4. Using SOAP

All the messages shown in the above figure are sent using SOAP (SOAP at one time stood for Simple Object Access Protocol. Now, the letters in the acronym have no particular meaning).

SOAP essentially provides the envelope for sending the web services messages. SOAP generally uses HTTP, but other means of connection may be used.

The next figure provides more detail on the messages sent using web services. At the left of the figure is a fragment of the WSDL sent to the directory. It shows a CustomerInfoRequest that requires the customer's account to object information. Also shown is the CustomerInfoResponse that provides a series of items on customer including name, phone, and address items.



```
<element name=”CustomerInfoRequest”>

 <element name=”account” type=”string”/>

<element>
<element name=”CustomerInfoResponse”>

  <element name=”name” type=”string”/>
  <element name=”phone” type=”string”/>
  <element name=”street1” type=”string”/>
  <element name=”street2” type=”string”/>
  <element name=”city” type=”string”/>
  <element name=”state” type=”string”/>
  <element name=”code” type=”string”/>
  <element name=”country” type=”string”/>
</element>
```

```
<element name=”CustomerInfoRequest”>

 <element name=”account” type=”string”/>

<element>
<element name=”CustomerInfoResponse”>

  <element name=”name” type=”string”/>
  <element name=”phone” type=”string”/>
  <element name=”street1” type=”string”/>
  <element name=”street2” type=”string”/>
  <element name=”city” type=”string”/>
  <element name=”state” type=”string”/>
  <element name=”code” type=”string”/>
  <element name=”country” type=”string”/>
</element>
```

```
<m:Get=CustomerInfo...>

 <account>1000</account>
<m:Get=CustomerInfo>
```

```
<m:Get=CustomerInfoResponse...>

  <name>Barry&Associates, Inc </name>
  <phone>123-456-789 </phone>
  <street1>1234 5th Avenue </street1>
  <street2></street2>
  <city>New York</city>
  <state>MN</state>
  <code>23146</code>
  <country>United States</country>
<m:Get=CustomerInfoResponse>
```

Directory
Service Description using WSDL
1
Query responses using WSDL
3
2
Directory Queries
XML service request based on WSDL
4
5
XML service response based on WSDL
Service Provider
Service Consumer

Figure 2.1.4: A typical WS message exchange.

At the right of this figure is a fragment of the WSDL being sent to the service consumer. This is the same fragment sent to the directory by the service provider. The service consumer uses this WSDL to create the service request shown above the arrow connecting the service consumer to the service provider. Upon receiving the request, the service provider returns a message using the format described in the original WSDL. That message appears at the bottom of the figure.

2.1.5. Using XML with WSDL

WSDL uses XML to define messages. XML has a tagged message format. This is shown in the above figure. The tag `<city>` has the value of "New York" and `</city>` is the ending tag indicating the end of the value of city. Both the service provider and service consumer use these tags. In fact, the service provider could send the data shown at the bottom of this figure in any order. The service consumer uses the tags and not the order of the data to get the data values.

2.1.6. WS-DAI

The Web Service Data Access and Integration (WS-DAI) family of specifications is an extension to the web services (WS) architecture that defines web service interfaces to data resources, such as relational or XML database. Developed by a working group of the Global

Grid Forum (GGF), it can be used both independently and integrated in a wider grid-architecture. In the specifications are included properties that can be used to describe the data service or the resource itself that is being provided, and the message patterns used to both access and execute queries and modifications. The representation of data is specified in a model-independent way and is realized in two inner specifications, the WS-DAIR and the WS-DAIX, that allow to define properties and operations for, respectively, relational and XML-based databases. Currently, it is anticipated that future specifications will be given to support access to RDF and object databases as well. Since the WS-DAI specifications define web services, the Web Service Description Language (WSDL) is used for the definition, while the messages are modelled using the Simple Object Access Protocol (SOAP). In a service-oriented grid-architecture environment, WS-DAI is likely to be used with further WS extension like WS-Security, WS-ResourceFramework and WS-AtomicTransaction, for security, resource identification and data transaction aspects, respectively.

WS-DAI specifies properties, messages and a set of interfaces that a web service could implement to become a data service that mediates the access to a particular data resource. An application interacting with this web service to access the data is called consumer.

Data resources. A data resource represents any system that can act as a source or sink of data and is logically divided in two categories by the way it is managed: externally or by service. An externally managed data resource models an existent DataBaseManagementSystem (DBMS) that actually stores the data: in this case, operations for carrying out database administration on the DBMS are not specified; instead, methods to access the data are provided. A service managed data resource is a data resource that is strictly related to the service-oriented middleware implementing WS-DAI specifications: its existence and lifetime is limited to this environment and it is accessedand totally managed through the DAIS specified functions.

Data resources are identified with names created according to the OGSAnaming schemes: these names could be both abstract and concrete. An abstract name is a location-independent and persistent identifier like a URI, while a concrete name specifies the physical location of the resource, using both a combination of service address plus abstract name or a reference defined following the WS-Addressing or WS-ResourceFramework specifications.

Data services. A web service that implements the interfaces specified by the WS-DAI extensions is defined as data service and exposes some properties that make possible to the consumer to interrogate the service, to determine whether the service is suitable for use in a given setting and obtain the information required to enable valid requests to be sent to the service.

These properties are specified by the WS-DAI and are organized, on request, in a specifications compliant XML document.

While extended properties can be introduced by further specialized sub-set of WS-DAI specifications, the general properties that all the data services must support are:

- **DataResourceAbstractName:** URI representing the abstract name of the data resource.
- **ParentDataResource:** If this resource was derived from another, this is the abstract name of the parent data resource.
- **DataResourceManagement:** An enumeration indicating if the data resource is ServiceManaged or ExternallyManaged .
- **DatasetMap:** A mapping between the QName of a message and the URI of a dataset type representing the result types supported for the messages.
- **ConfigurationMap:** A mapping between the Qname of a message and the URI of an expression language (example XPath Version 1.0 or 2.0).

- **LanguageMap:** A mapping between the QName of a message and the URI of an expression language.
- **DataResourceDescription:** A human readable description of a data resource.
- **Readable/Writeable:** These properties indicate whether or not the data resource is able to be read from or written to by the data service.
- **ConcurrentAccess:** Has the value true if a data service is able to process more than one message concurrently, otherwise it has the value false.
- **TransactionInitiation:** Describes under what circumstances a transaction is initiated in response to messages. The values are as follows:
  - **NotSupported:** does not support transactions;
  - **Automatic:** transaction initiated automatically for the duration of each message;
  - **Manual Transaction:** context under control of the consumer, for example using an existing transaction specification, such as WSAtomicTransaction.
- **TransactionIsolation:** Describes how transactions behave with respect to other ongoing transactions.
- **ChildSensitiveToParent/ParentSensitiveToChild:** Indicates whether a parent or child data resource is sensitive to changes made to the other.

Access. The WS-DAI relies, for data access operations, on the existing facilities of the integrating DBMS: A data service specified by WS-DAI could be seen as a wrapper for the DBMS in the middleware. The specifications support two patterns for obtaining the results of requests directed at a data resource, referred to as direct data access and indirect data access.

Direct data access follows the typical web service type of interaction, where a consumer expects the data or status of a query in the response to a request. For direct data access, the WS-DAI specification only defines a single query-language-independent message (Generic Query), and a template for realizations to follow in defining language-dependent operations.

Indirect data access essentially implements the factorypattern, whereby the result of a request is not returned directly to the user, but rather made available as a Service Managed Data Resource in its own right, for access through a data service. The consumer thus gets a reference in the response message through which the data may be accessed. The WSDAI specification does not define any generic indirect access operations, but it defines a template that must be implemented by WS-DAI specialized extensions.

WS-DAI family specifications provide several benefits to the development of databases through exposing a standard way to access the data, thanks to the WS definition layer. This way an extension of database availability is easy, cutting the needs of client-side software deployment and the possibility of integrations with other web service standards enabling extended features like security, transaction management and data movement. The major benefit, obviously, is in a service-oriented grid middleware, where a systematic and composite access to data could really result in improving cost effectiveness of the system.

2.1.7. References

Mario Antonioletti, Amy Krause, Norman W. Paton, Andrew Eisenberg, Simon Laws, Susan Malaika, Dave Pearson, Jim Melton The WS-DAI Family of Specifications forWeb Service Data Access and Integration

2.2. Open Grid Services Architecture

Grid systems and applications aim to integrate, virtualize and manage resources and services within distributed, heterogeneous, dynamic "Virtual Organizations" (VO) [Grid Anatomy, Grid

Physiology]. The realization of this goal requires the disintegration of the numerous barriers that normally separate different computing systems within and across multiple organizations, so that computers, application services, data and other resources can be accessed as and when required, regardless of physical location. To the realization of this grid vision, standardization is a key requirement, so that the different components that make up a modern computing environment can be discovered, accessed, allocated, monitored, accounted for, billed for and, more in general, managed as a single virtual system, even if they are provided by different vendors or operated by different organizations. Standardization is fundamental to create interoperable, portable, reusable components and systems. Moreover, standardization can also contribute to the definition of secure, robust and scalable grid systems, by facilitating the use of good practices.

The Open Grid Services Architecture (OGSA) is a service-oriented architecture, developed within the Global Grid Forum (GGF) to satisfy this need for standardization, by defining a set of core capabilities and behaviours that address key concerns in grid systems. These concerns include issues such as discovering of services, negotiation and monitoring of service level agreements, management of membership and communication within VOs, hierarchical collection of services so as to deliver reliable and scalable service semantics, integration of data resources into computations, monitoring and management of services collections.

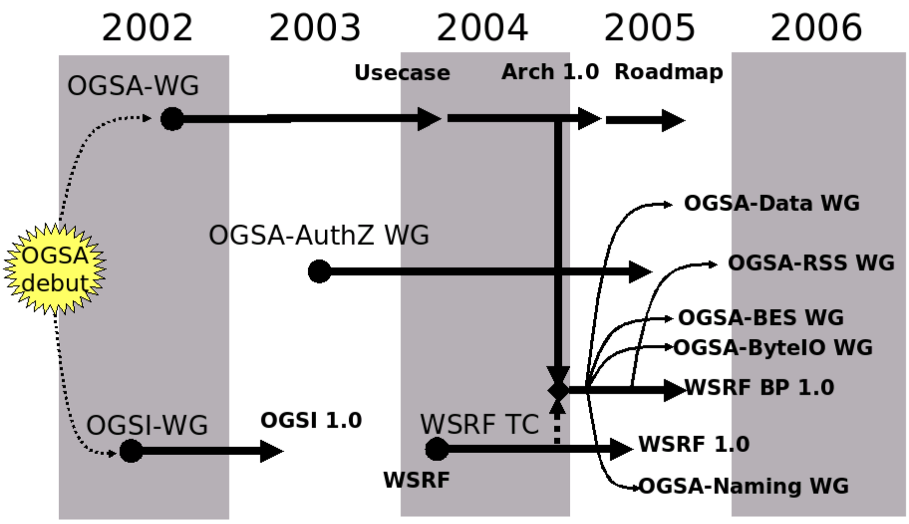


Figure 2.2.1: OGSA and WSRF history.

OGSA is based on several other web service technologies, notably WSDL and SOAP, but it aims to be largely agnostic in relation to the transport-level handling of data.

Briefly, OGSA is a distributed interaction and computing architecture based around services, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. OGSA has been described as a refinement of the emerging web services architecture, specifically designed to support grid requirements. OGSA has been adopted as a grid architecture by a number of grid projects including the Globus Alliance.

2.2.1. OGSA Features

According to the OGSA Roadmap document [REF], we have to distinguish three different areas, which are all essential to maintain coherence between OGSA and grid standards:

- An architectural process, managed by the GGF's OGSA Working Group, which collects requirements and maintains a set of informational documents that describe the architecture.
- A set of normative specifications and profiles that document the precise requirements for a conforming hardware or software component. In particular, a specification documents the technical requirements, typically including interfaces, protocols and behaviors, for a conforming hardware or software component. An OGSA Profile identifies a set of broadly adopted normative technical specifications that collectively capture current understanding of what software must do to operate and manage interoperable grid environments.
- Software components that adhere to the OGSA specifications and profiles, enabling deployment of grid solutions that are interoperable even though they may be based on implementations from multiple sources.

2.2.2. OGSA Requirements

The definition of OGSA 1.5 [REF] is driven by a set of functional and non-functional requirements, which resulted from a selected set of use-cases [OGSA Use Cases] [OGSA Use Cases Tier 2], covering infrastructure and application scenarios for both scientific and commercial areas.

These use-cases include Commercial Data Centre, Severe Storm Modelling, Online Media and Entertainment, National Fusion Collaboratory, Service-Based Distributed Query Processing, Grid Workflow, Interactive Grids, Grid Lite, Reality Grid, etc.

Functional and non-functional requirements that appear both important and broadly relevant are the following:

- **Interoperability** and support for dynamic and heterogeneous environments: resource virtualization, common management capabilities, resource discovery and query, standard protocols and schemas
- **Resource sharing** across organizations: global name space, metadata services, site autonomy, resource usage data
- **Optimization**
- **Quality of Service** assurance: SLA (Service Level Agreement), service level attainment, migration
- **Job execution:** support for various job types, job management, scheduling, resource provisioning
- **Data services:** policy specification and management, data storage, data access, data transfer, data location management, data update, data persistency, data federation
- **Security:** authentication and authorization, multiple security infrastructures, perimeter security solutions, isolation, delegation, security policy exchange, intrusion detection, protection and secure logging
- **Administrative cost reduction:** policy-based management, application contents management mechanisms, problem determination mechanisms
- **Scalability:** management architecture scaling, High-throughput computing mechanisms
- **Availability:** disaster recovery mechanisms, fault management mechanisms
- **Ease of use** and extensibility

2.2.3. Capabilities

OGSA is intended to facilitate the seamless use and management of distributed, heterogeneous resources. The utility provided by such an architecture is realized as a set of capabilities. Figure 2.2.2 shows the logical, abstract, semi-layered representation of some of these capabilities. Three major logical and abstract tiers areenvisioned in this graphical representation.

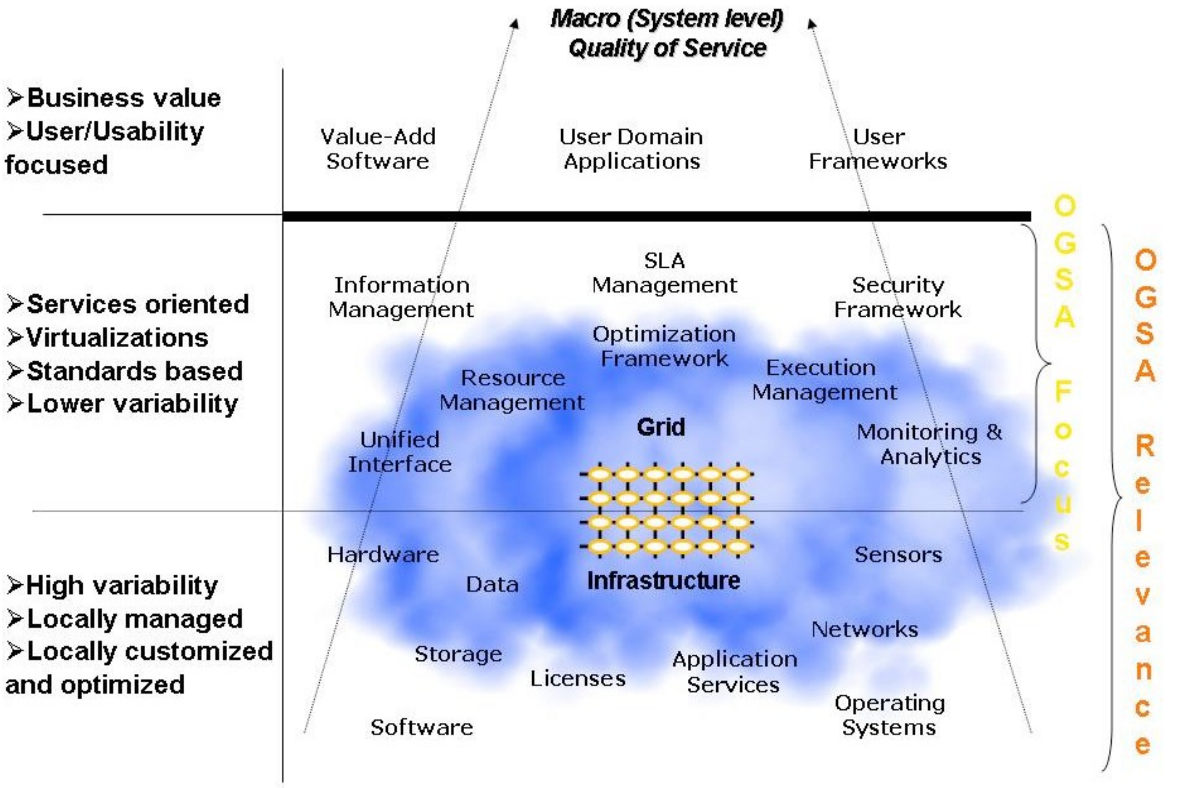


Figure 2.2.2: OGSA capabilities.

The first (bottom) tier depicts the base resources, which are the resources supported by some underlying entities or artifacts (physical or logical) and that have some relevance outside of the OGSA context.

The second (middle) tier represents a higher level of virtualization and logical abstraction. The virtualization and abstraction are directed toward defining a wide variety of capabilities that are relevant to OGSA grids. It should be noted that the capabilities shown in the diagram are only a subset of the complete set of OGSA capabilities.

At the third (top) tier in the logical representation, there are the applications and other entities that use the OGSA capabilities to realize user and domain oriented functions and processes, such as business processes.

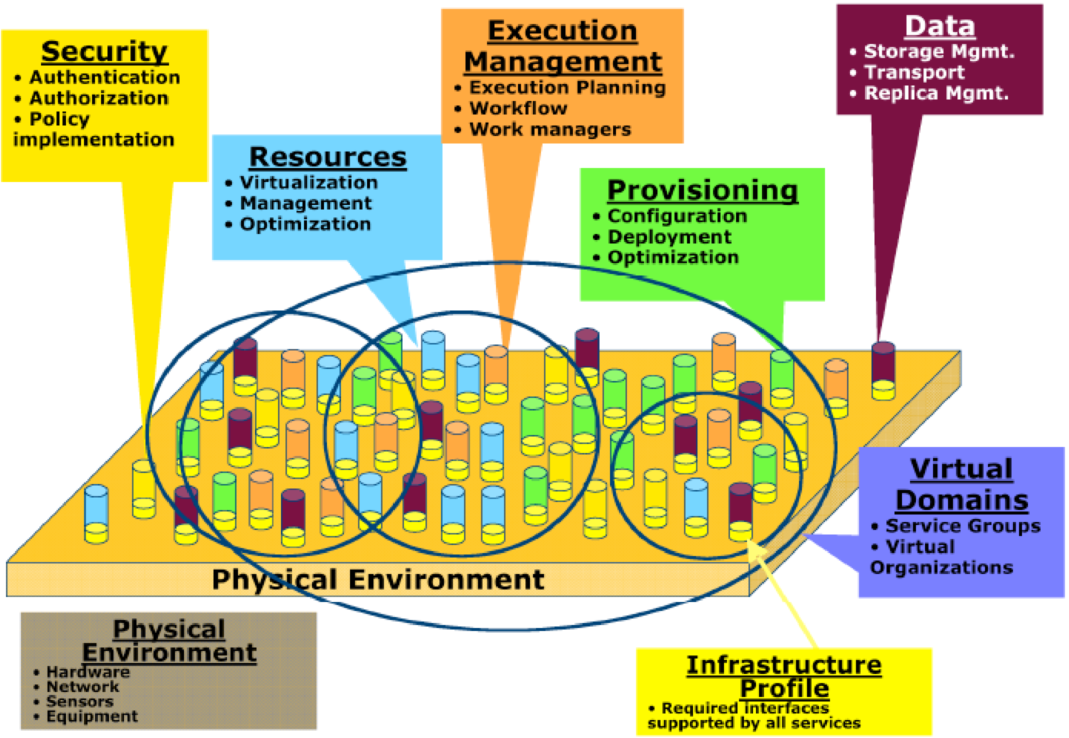


Figure 2.2.3: OGSA framework.

Figures 2.2.3 and 2.2.4 show the OGSA framework. In particular, cylinders represent individual services, built on web services standards with semantics, additions, extensions and modifications that are relevant to grids.

2.2.4. Infrastructure Services

OGSA capabilities share and build on a number of common components. The primary assumption is that OGSA builds on, and is contributing, to the development of, the collection of technical specifications that form the emerging Web Services Architecture. Indeed, OGSA can be viewed as a particular profile for the application of core WS standards. This choice of web services as an infrastructure and framework means that OGSA assumes that systems and applications are structured according to service oriented architecture principles and the service interfaces are defined by the Web Services Description language (WSDL). Moreover, XML is used for description and representation purposes and SOAP is the primary message exchange format for OGSA services. Since WS standards, as currently defined are not able to meet all grid requirements, OGSA architects have been involved in the definition of WSDL 2.0, too. In addition to "Web Services Foundation", the other infrastructure services are naming (naming scheme and naming policy), security services, representing state, notification, transactions, orchestration, profiles to ensure interoperability.

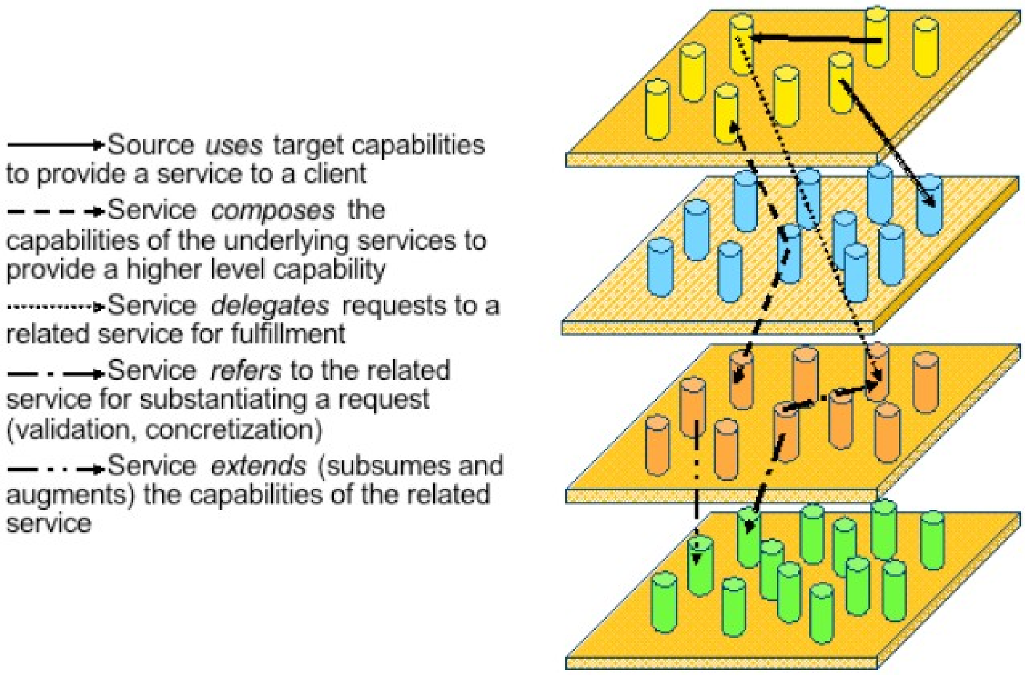

Source *uses* target capabilities to provide a service to a client
Service *composes* the capabilities of the underlying services to provide a higher level capability
Service *delegates* requests to a related service for fulfillment
Service *refers* to the related service for substantiating a request (validation, concretization)
Service *extends* (subsumes and augments) the capabilities of the related service

Figure 2.2.4: OGSA service dependencies.

2.2.5. Execution Management Services

These services are concerned with the problems of instantiating and managing, to completion, units of work, such as OGSA applications or legacy (non-OGSA) applications. The issues to deal with, when an application needs a service (e.g. a cache service), range from the choice between the use of an existing service and the creation of a new one, the placement of the new service, its configuration, the provision of the adequate resources, the service agreements to establish and the required agreements. More formally, Execution Management Services (EMS) address problems with executing units of work, including their placement, provisioning and lifetime management. Therefore, it is necessary to find execution candidate locations, select execution location, prepare for execution, initiate the execution and, finally, manage the execution.

The solution consists of a set of services that decompose the EMS problem into multiple, replaceable components. There are three broad classes of EMS services:

- Resources, which model processing, storage, executables, resource management and provisioning.
- Job management.
- Resource selection services that collectively decide where to execute a unit of work.

EMS interact with other parts of OGSA, such as the deployment and configuration service, naming, information service, monitoring, fault-detection and recovery services, logging.

2.2.6. Data Services

These services concern with the management of, access to and update of data resources, along with the transfer of data among resources. They also provide the capabilities needed to manage the metadata that describes this data and, in particular, the origin of the data itself.

The heterogeneous nature of the grid implies that many different data types must be supported, such as flat files, streams, DBMS, catalogues, derivations, data services that are data resources for other services (e.g. sensor devices, measurement instruments, programs). Figure 2.2.5 illustrates the basic entities of the data architecture. Resources are managed by services that may have interfaces for data access, for acting as sources or sinks for data transfer operations and for describing the resources via properties. Since some resources are storage-based (e.g. file-systems, database, etc.) the architecture also indudes interfaces for managing that storage.

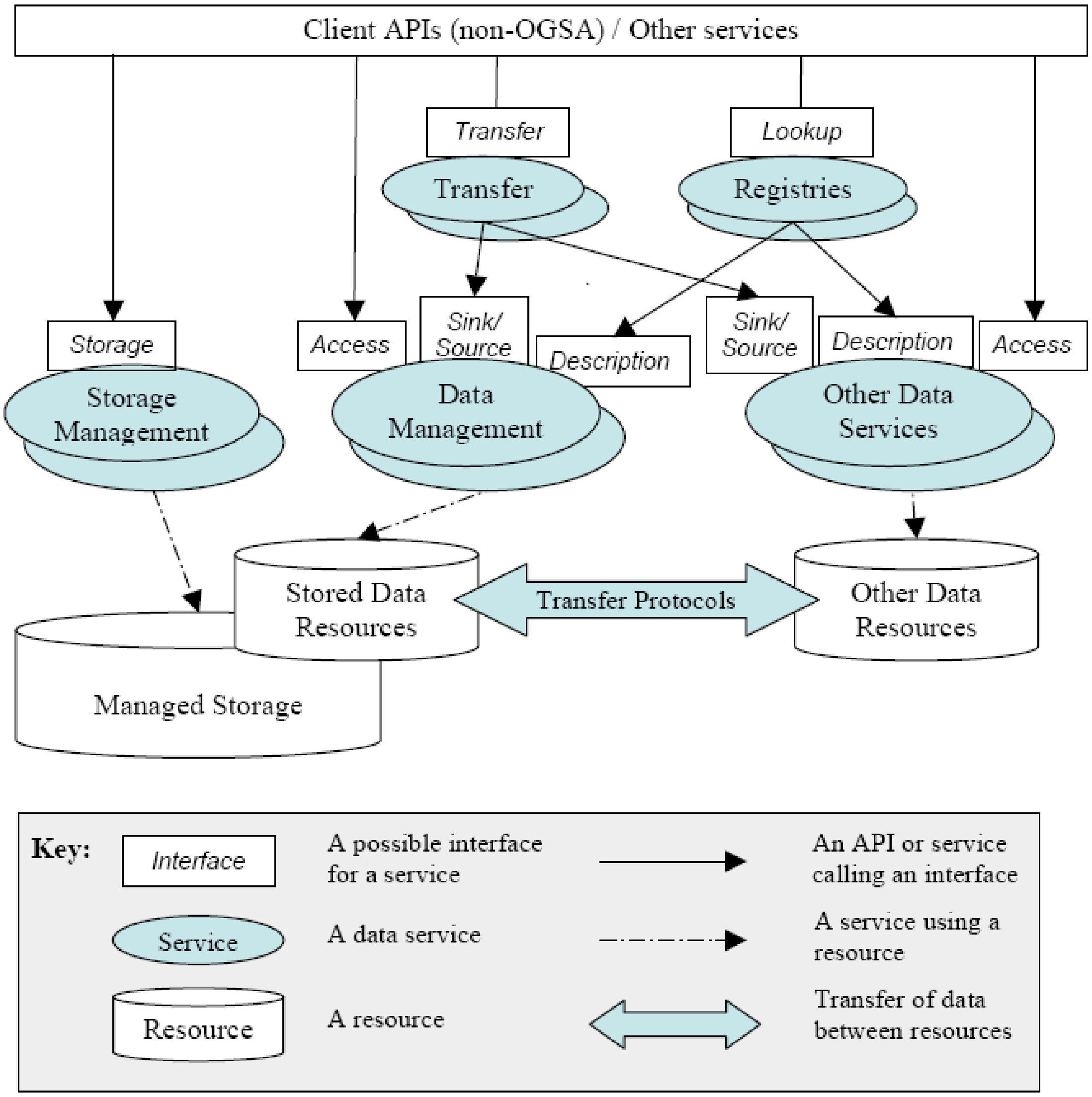


Figure 2.2.5: Basic entities of the OGSA data architecture.

The architecture fits with non-OGSA data resources and access mechanisms, thus enabling OGSA functionality to augment existing and future data infrastructures. Indeed, OGSA guarantees transparency and virtualization, client API, extensible data types support and operation.

The functional capabilities provided by the OGSA data services include: data transfer, storage management, simple access, queries, federation, location management, update, transformation, security mapping extensions, resource and service configuration, metadata catalogues, data discovery, provenance.

2.2.7. Resource Management Services

Resource management performs several forms of management on resources in a grid. In an OGSA grid, there are three types of management that involve resources:

- Management of the physical and logical resources themselves (e.g. rebooting a host, setting VLANs on a network switch, etc.).
- Management of the OGSA grid resources exposed through service interfaces (e.g. resource reservation, job submission and monitoring, etc).
- Management of the OGSA grid infrastructure, exposed through its management interfaces (e.g. monitoring a registry service).

Different types of interfaces realize the different types of management in an OGSA grid. These interfaces can be classified into three levels, as shown in table 2.2.6.

Type of Management	**Level of Interface**	**Interface**
management of the physical and logical resources	resource level	CIM/WBEM, SNMP, etc.
	infrastructure level	WSRF, WSDM, etc.
management of OGSA grid resources	OGSA functions level	functional interface
mgment. of OGSA grid infrastructure		specific manageability interfaces

Table 2.2.6: OGSA interface types.

2.2.8. Security Services

These services facilitate the enforcement of the security-related policy within a VO. The functional capabilities and the corresponding security services are the following:

- authentication
- identity mapping
- authorization
- credential conversion
- audit and secure logging
- privacy

2.2.9. Self-Management Services

Self management services aim to support service-level attainment for a set of services (or resources, depending on the taxonomy), with as much as automation as possible to reduce the costs and complexity of managing the system. In an operational environment, it is often necessary to control various aspects of the behaviour of a solution component in a manner that cannot be determined a priori by the component developer.

The collection of attributes needed for various stages of self-management include service level management, policy, service level manager model.

The functional capabilities, based on mechanisms for self-configuration, self-healing and self-optimizing are the following:

- service level management (monitoring, analysis and projection, action)
- policy and model management
- entitlement
- planning
- capability management
- provisioning
- analytics

2.2.10. Information Services

The following functional capabilities have been defined:

- discovery
- message delivery
- logging
- monitoring
- general information and monitoring services

As of late 2006 an updated version of the OGSA architecture document and several associated documents have been published, including the first of several planned normative documents, the OGSA WSRF Basic Profile, Version 1.0. Development of conformant software is expected to follow rapidly once a critical mass of normative documents havebeen published.

2.2.11. OGSA-DAI

The "Open Grid Service Architecture — Data Access and Integration" (OGSA-DAI) is a project, extending the Open Grid Service Architecture (OGSA), aimed to develop an infrastructure capable to provide a uniform way of collecting and manipulating data.

This project has started with the intention to realize a reference implementation for the WS-DAI standards elaborated by Database Access and Integration Services Work Group (DAIS-WG) from the Open Grid Forum (OGF). This work is based on the specifications released in 2003 and is currently occurring in parallel with the development of WS-DAI, with several reciprocal influences.

Being based on the web service technology for the presentation layer, OGSA-DAI achieves a high degree of programming languages neutrality, allowing the developer to extend the core web-service for application-specific purposes.

Currently, two popular specifications are supported: Web Services Inter-operability (WS-I) and the Web Services Resource Framework (WSRF).

The two specifications have been implemented in versions compatible with other widely used implementations, like: UK OMII for the WS-I, and Globus Toolkit for the WSRF.

Architecture and functionalities. Rather than creating a new grid enabled database management system, OGSA-DAI follows the approach of integrating some of the most widely used database technology through the use of Data Resource abstraction: this layer is realized by two components, the Data Resource Accessor and the Data Service Resource, that make possible to expose the functionality of the physical resource in the OGSA-DAI environment.

Data Resource Accessors take care of performing access and operation on the physical resources: to maintain a high degree of portability they are currently written in Java programming language and currently support relational (MySQL, Microsoft SQL Server, Oracle, IBM DB2, PostGres, HSQL), XML (eXist, Xindice) and file system based (SwissPROT, OMIM, text, binary) databases.

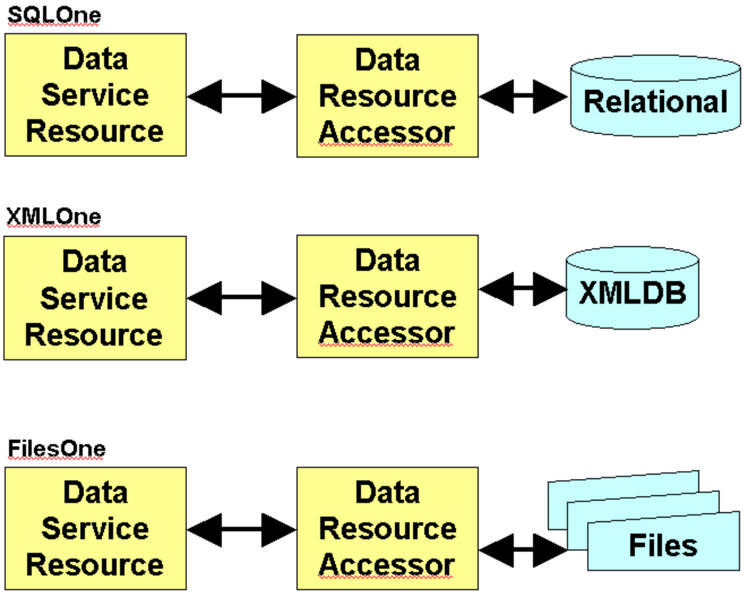


Figure 2.2.7: Database support in OGSA-DAI.

The duty of Data Service Resources, that compose the core of the OGSA-DAI functionalities, is to manage the activity session, to perform them, to manage the aggregation and delivery of the data and to expose the information about the managed resource.

Another key aspect of OGSA-DAI infrastructure is the Data Service component that manages the various Data Service Resources and performs the intermediation between them and Clients.

As mentioned before, operations to be performed on data are called Activities in the OGSA-DAI architecture: they could be relative to the specific database (relational, XML based, etc.) or realize operations like data transformation, enabling the use of various formats for data view presentation, data compression/decompression, resource construction using several patterns (DataServiceResource, MultiResource for aggregate data from several physical databases, FactoryResource) and destruction, data delivery, using several technologies, like HTTP or SOAP, to different consumers like clients or other OGSA-DAI structures or, finally, data storage using a technology like GridFTP.

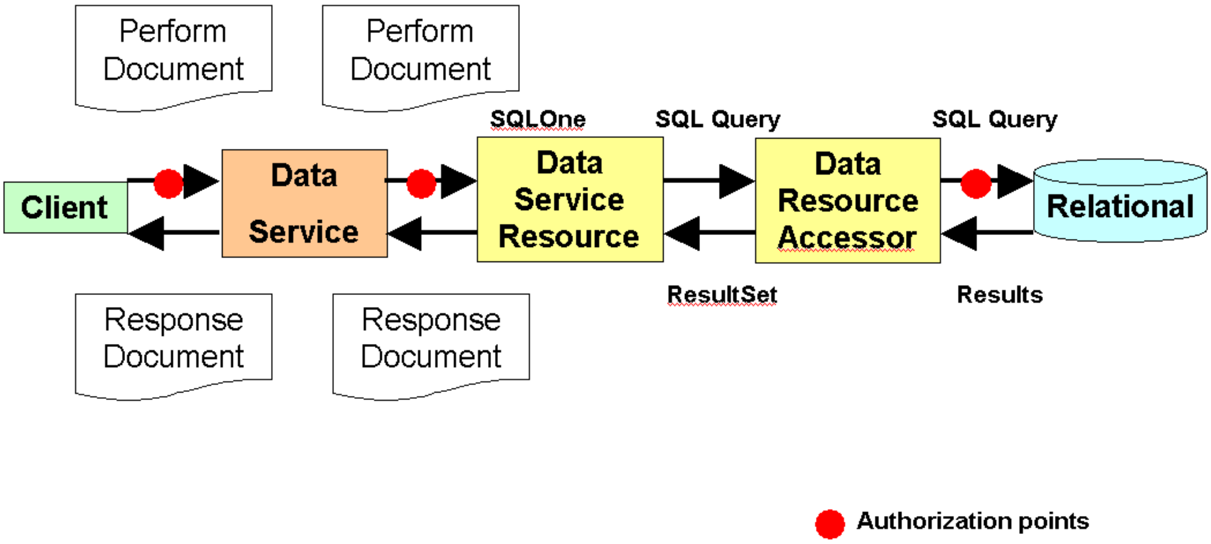


Figure 2.2.8: Performing activities by document exchange.

OGSA-DAI, as noted in the previous figure, also offers a complete layer that enforces security policies at several levels in the architecture: access to data can be regulated using the authorization framework, the execution of activities can be controlled and restricted through a permission mechanism and security techniques can be applied to both message-level and data transport-level.

Extensibility. The current implementation of OGSA-DAI models the introduced abstractions using Java technology that allows developers to extend the behavior and functionalities of the OGSA-DAI parts: new Data Resource Accessors can be developed to support other databases, the available activities can be extended to perform application-specific operations on data, and the security layer can be adapted to model particular application scenarios' needs.

OGSA-DAI offers an out-of-the-box solution to integrate databases in the grid environment using the core functionalities offered or extending them to meet the application scenarios. Being based on WS-DAI specifications, it offers both coherence of design and transparence and portability characteristics, while allowing an easy, time-saving way to integrate the support for data manipulation and management in grid based projects.

2.2.12. References

[1] The OGSA-DAI project, http://www.ogsadai.org.uk/

[2] The Open Grid Forum, http://www.ogf.org/

[3] Open Grid Forum—Database Access and Integration Services Working Group, http://forge.gridforum.org/projects/dais-wg/

[4] Web Service Interoperability Organization, http://www.ws-i.org/

[5] Web Service Resource Framework, http://www.globus.org/wsrf/

[6] The Open Middleware Infrastructure Institute UK, http://www.omii.ac.uk/

[7] Mario Antonioletti, Malcolm Atkinson, Rob Baxter,Andrew Borley, Neil P. Chue Hong, Brian Collins,Neil Hardman,Alastair C. Hume,Alan Knox,Mike Jackson,Amy Krause, Simon Laws,James Magowan, NormanW. Paton,Dave Pearson,Tom Sugden, PaulWatson and Martin Westhead The design and implementation of Grid database services in OGSA-DAI

2.3. Resource State Management

Generic resource management for instrument resources is not greatly different than that for computing and storage resources, when it comes to typical management such as brokering, scheduling, monitoring and accounting. However, state management is of particular importance when the resources are instruments. While computing and storage resources can be handled in a trial-and-error fashion (making requests and handling failures), this is not the case with instruments. Instruments are much more sensitive to poor operation, and prone to hardware failure if they are stressed over their limits. As such, we need to be able to verify their state under all circumstances, and refrain from operating them beyond their recommended limits. The GridCC project middleware, described later in this document, has implemented a state machine within the "Instrument Element"abstraction, exactly for this reason.

Web services are the technology of choice for Internet-based applications with loosely coupled clients and servers. That makes them the natural choice for building the next generation of grid-based applications. However, web services do have certain limitations. In fact, plain web services are unable to manage state of resources. The "Web Services Resource Framework" (WSRF) improves several aspects of web services to make them more adequate for grid applications. Essentially, we need to define conventions for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. The WS-Resource Framework defines these conventions and does so within the context of established web services standards. The WSRF supersedes the "Open Grid Services Infrastructure" (OGSI), which defined non-standard WSDL extensions for this reason.

In this section we'll take a brief look at the different parts of the WSRF specification. However, before doing that, we need to take a close look at the main improvement in WSRF: Statefulness.

2.3.1. State Management

Plain web services are usually stateless (even though, in theory, there is nothing in the web services Architecture that says they can't be stateful). This means that the web service cannot "remember" information, or keep state, from one invocation to another.

The fact that web services do not keep state information is not necessarily a bad thing. There are plenty of applications which have no need whatsoever for statefulness. However, grid applications do generally require statefulness. So, we would ideally like our web service to somehow keep state information.

The resource approach to statefulness. Giving web services the ability to keep state information while still keeping them stateless seems like a complex problem. Fortunately, it's a problem with a very simple solution: simply keep the web service and the state information completely separate.

Instead of putting the state in the web service (thus making it stateful, which is generally regarded as a bad thing) we will keep it in a separate entity called a "resource", which will store all the state information. Each resource will have a unique key, so whenever we want a stateful interaction with a web service we simply have to instruct the web service to use a particular resource. A web service can have access to more than one resource.

One might wonder how exactly does the client specify what resource must be used. A URI might be enough to address the web service, but how do we specify the resource on top of that? There are actually several different ways of doing this. As we'll see later on, the preferred way of doing it is to use a relatively new specification called WS-Addressing which provides a more versatile way of addressing web services (when compared to plain URIs).

A pairing of a web service with a resource is called a WS resource. The address of a particular WS-Resource is called an endpoint reference, in WS addressing terminology.

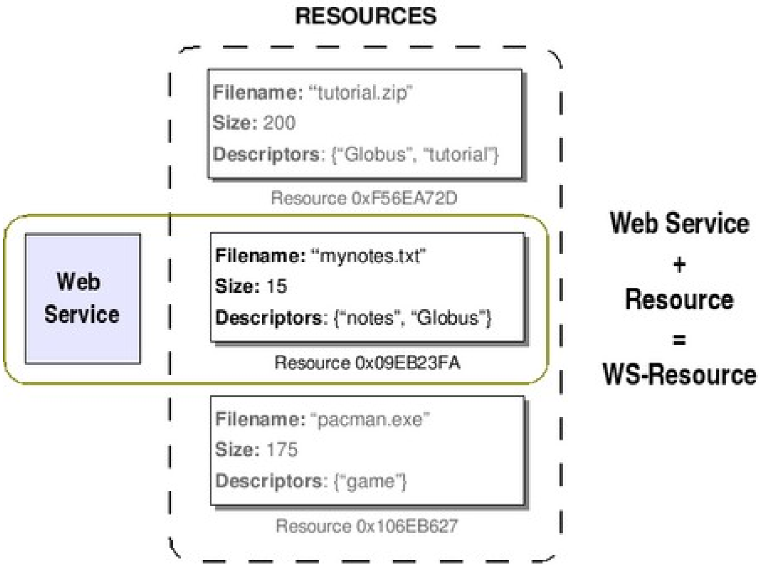


Figure 2.3.1: WS resource.

Separation of service and stateful resources. Any discussion of state in a web or grid service brings up the question, "Do we really want all the overhead and restrictions of placing state directly within such a service?" The immediate answer is no. Placing that entire encumbrance on web services standards would do a mighty disservice to the hundreds and thousands of existing web services that neither need nor use statefulness as part of their modus operandi. For example, many web services are mere interfaces for querying information about inventory levels, as in the popular Amazon.com web service. Request goes in, response comes out, and this is the only requirement.

So there is no need to embed state directly into a grid or web service. A better approach is to keep grid/web services interfaces stateless, and let them interact with separate stateful resources. That way, one's grid/web service can be restarted and it will reconnect with any of the outside components that provide state information.

This is precisely what the underlying WSRF tools do. They allow a grid or web service to extract and modify state information, regardless of the kind of resource being used to store that information — XML documents, relational database tables, object code in a server session or EJB, and so on. This stateful information can then be uniquely identified and inserted into the message stream used by the service, just like any other part of the XML documents. It's important to note here that what we are dealing with is not the stateful resource itself, but a reference to that stateful resource.

Because this reference has a unique identifer, it allows us to execute such actions as immediate or scheduled destruction of that reference — which in turn disallows further messages in that sequence from being acted upon.

2.3.2. WSRF: The Specification

WSRF is a family of specifications published by OASIS. Major contributors include the Globus Alliance and IBM. As previously explained, it provides a set of operations that web services may implement to become stateful; web service clients communicate with resource services which allow data to be stored and retrieved. When clients talk to the web service they include the identifier of the specific resource that should be used inside the request, encapsulated within the WS-Addressing endpoint reference. This may be a simple URI address, or it may be complex XML content that helps identify or even fully describe the specific resource in question.

Alongside the notion of an explicit resource reference comes a standardized set of web service operations to get/set resource properties. These can be used to read and perhaps write resource state, in a manner somewhat similar to having member variables of an object alongside its methods. The primary beneficiaries of such a model are management tools, which can enumerate and view resources, even if they have no other knowledge of them. This is the basis for Web Services Distributed Management (WSDM).

The WSRF specification includes mechanisms to describe views on the state, to support management of the state through properties associated with the web service, and to describe how these mechanisms are extensible to groups of web services. It defines the means by which:

- Web services can be associated with one or more stateful resources (named, typed, state components).
- Service requestors access stateful resources indirectly through web services that encapsulate the state and manage all aspects of web service based access to the state.
- Stateful resources can be destroyed, through immediate or time based destruction.
- The type definition of a stateful resource can be associated with the interface description of a web service to enable well-formed queries against the resource via its web service interface.
- The state of the stateful resource can be queried and modified via web service message exchanges.
- Endpoint references to web services that encapsulate stateful resources can be renewed when they become invalid, for example due to a transient failure in the network.
- Stateful resources can be aggregated for domain-specific purposes.

2.3.3. Component Specifications

WS-Resource defines a WS-Resource as the composition of a resource and a web service through which the resource can be accessed.

WS-ResourceProperties describes an interface to associate a set of typed values with a WS-Resource that may be read and manipulated in a standard way. This defines how the data associated with a stateful resource can be queried and changed using web services technologies. This allows a standard means by which data associated with a WS-Resource can be accessed by clients. The declaration of the WS-Resource's properties represents a projection of or a view on the WS-Resource's state. This projection represents an implied resource type which serves to define a basis for access to the resource properties through web service interfaces.

WS-ResourceLifetime describes an interface to manage the lifetime of a WS-Resource. This defines two ways of destroying a WS-Resource: immediate and scheduled. This allows designers flexibility to design how their web services applications can clean up resources no longer needed.

WS-BaseFaults describes an extensible mechanism for rich SOAPFaults. This defines an XML Schema type for a base fault, along with rules for how this fault type is used by web services. A designer of a web services application often uses interfaces defined by others. Managing faults in such an application is more difficult when each interface uses a different convention for representing common information in fault messages. Support for problem determination and fault management can be enhanced by specifying web services fault messages in a common way. When the information available in faults from various interfaces is consistent, it is easier for requestors to understand faults. It is also more likely that common tooling can be created to assist in the handling of faults.

WS-ServiceGroup describes an interface for operating on collections of WS-Resources. This defines a means by which web services and WS-Resources can be aggregated or grouped together for a domain specific purpose. In order for requestors to form meaningful queries against the contents of the ServiceGroup, membership in the group must be constrained in some fashion. The constraints for membership are expressed by intension using a classification mechanism. Further, the members of each intension must share a common set of information over which queries can be expressed.

2.3.4. Related Specifications

WS-Notification is another collection of specifications that, although not a part of WSRF, is closely related to it. This specification allows a web service to be configured as a notification producer, and certain clients to be notification consumers (or subscribers). This means that if a change occurs in the web service (or, more specifically, in one of the WS-Resources), that change is notified to all the subscribers (not all changes are notified, only the ones the web services programmer wants to).

WS-Addressing. As mentioned before, the WS-Addressing specification provides us a mechanism to address web services which is much more versatile than plain URIs. In particular, we can use WS-Addressing to address a web service + resource pair (a WS-Resource).

The phrase "implied resource pattern" describes the way WS-Addressing is used to associate a stateful resource with the execution of message exchanges implemented by a web service. A WS-Addressing EndpointReference that follows the implied resource pattern must include a ReferenceProperties child element that identifies the resource to be associated with the execution of all message exchanges performed using this EndpointReference. A web services message that follows the implied resource pattern must be sent to a web service referred to by an EndpointReference that follows the implied resource pattern, and must include the ReferenceProperties information from that EndpointReference, as specified by WS-Addressing. A web service that follows the implied resource pattern must use the ReferenceProperties information from a message that follows the implied resource pattern in order to identify the resource to associate with the execution requested by that message.

2.3.5. The WSRF Specification Compared to OGSI

OGSI (Open Grid Services Infrastructure) is an old standard, proposed by the Global Grid Forum in 2003. It has been superseded by theWSRF Specification.

2.3.6. Value of WSRF to Customers and Software Developers

The single most valuable aspect of WSRF is that it effectively completes the convergence of the web service and grid computing communities.

Grid customers will benefit from the increased quality and variety of web service environments supporting the capabilities required for grid computing. The grid developer, likewise, will benefit from a wider selection of development tools.

Web services customers will benefit from the availability of implementations of the powerful resource management primitives embodied in WSRF.

2.3.7. WSRF Relations to Other Web Services Standards

WSRF specifications build directly on core web services standards, in particular WSDL, SOAP, and XML, and exploit capabilities provided by WS-Addressing. WSRF specifications introduce mechanisms that we expect to be applicable to emerging specifications such as those being developed within the GGF Data Access and Integration Services (DAIS) working group and the OASIS Web Services Distributed Management (WSDM) technical committee.

2.3.8. Implementations

Implementing the basic property get/set semantics of WSRF resources is relatively simple. The hardest problem is probably returning faults as WSRF Base Faults where the specification requires it, because SOAP stacks themselves prefer to raise SOAPFault faults. Managing resource lifetimes is harder, but this is optional, as is WS-Notification, which is the hardest to test.

- The Globus Toolkit version 4 contains Java and C implementations of WSRF; many other Globus tools have been rebuilt around WSRF.
- WebSphere Application Server version 6.1 provides a WSRF environment which supports both simple and clustered, highly available WSRF endpoints.
- The Apache Foundation has a Muse 2.0 project which is a Java-based implementation of the WSRF, WS-Notification, and WSDM specifications.
- WSRF::Lite is a perl-based implementation that makes exclusive use of the Address element of the endpoint reference, thus making WS-Resources identifiable via URIs. In addition, WSRF::Lite provides a mapping of HTTP verbs to WSRF operations, making it possible to use WS-Resources in a REST architectural style.
- WSRF.NET is a .NET based project about WSRF specs from a research team of the University of Virginia.

2.3.9. Conclusions

We have presented the WS-Resource framework, a set of web service specifications and conventions designed to standardize representation of, and access to, stateful resources in a distributed environment. This framework identifies and standardizes the patterns by which state is represented and manipulated, so that a web service can describe the stateful resources to which it provides access, and a service requestor can discover the type of this pairing of web service and stateful resource ("WS-Resource") and use standardized operations to read, update, and query values of its state, and to manage its lifecycle. The definition of the WS-Resource framework facilitates the construction and use of interoperable services, by making it possible for different service providers and service consumers to describe, access, and manage their stateful resources in standard ways. Equally importantly, the framework introduces support for stateful resources without compromising the ability to implement web services as stateless message processors. The framework also addresses issues of renewable references, grouping, notification, and fault reporting.

2.3.10. References

http://en.wikipedia.org/wiki/WS-Addressing

http://www.globus.org/wsrf/

http://en.wikipedia.org/wiki/Web_Services_Resource_Framework

http://www.service-architecture.com/web-services/articles/web_services_resource_framework_wsrf.html

http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s03.html

http://www-128.ibm.com/developerworks/grid/library/gr-wsrf.html

http://www-128.ibm.com/developerworks/webservices/library/specification/ws-resource/ws-wsrfpaper.html

2.4. Data Management

Instruments, especially expensive ones with large scale in size and high performance in observation resolution, produce large amounts of data, and a large number of remote instrumentation applications require access to large amounts of data with varied quality of service requirements and network conditions. For such applications, data sources (the instruments) generate a large amount of data, and users who require access to the data are geographically dispersed in many cases. The data management, especially data acquisition, becomes the key issue that affects the overall performance of applications. Data movement is a fundamental operation among storage systems and between programs and data storage, and provides the foundation for replication, caching, and bulk data access. Replication acts as a high-level service that is built using basic data movement function. It creates replicas to reduce access latency and network bandwidth consumption, maintain local control over transient and necessary data, and improve reliability and load balancing.

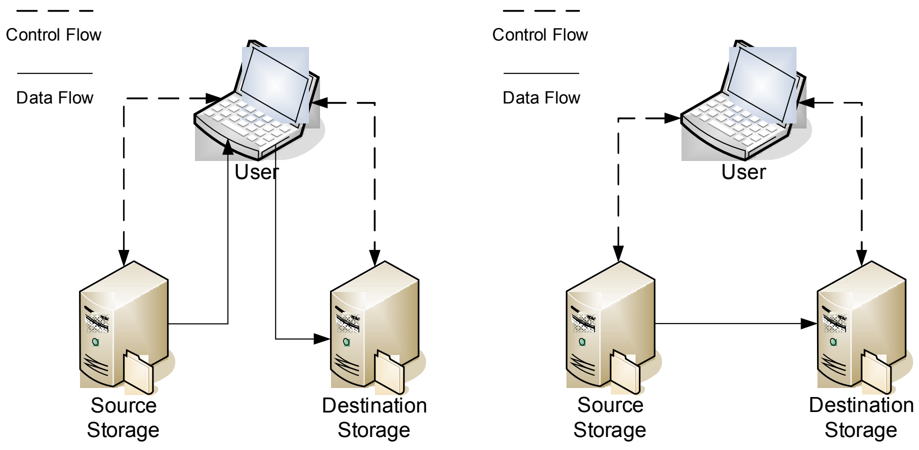
2.4.1. Data Movement Basic

Data movement is the basic service for data management, and has been widely studied. A variety of protocols manage the movement of data among data sources. Among those protocols, two types of protocols can be categorised, low level transfer and higher level transfer.

Low Level Transfer in Grid Environment. Low level transfer focuses on actually shifting data between machines in the grid environment. Files, especially bulk files, are frequently required to transmit among remote servers. The Internet provides end-to-end data transmission protocols such as FTP and HTTP. However, it is inefficient to make the data flow via the machine orchestrating the transfer for the following reasons:

1. The management machine might be a low performance machine (e.g. laptop or PDA) on a poor network connection;
2. A triangle transmission might happen where excessive transmissions have been done to pass through the orchestrating machine.

In order to solve the drawback, tools such as GridFTP, supporting the third party transfers between remote sites, have been proposed in addition to traditional file transfer protocols. As illustrated in figure 2.4.1, by enabling the third party transfer function, control flows are decoupled from data that follow. Control channels are set between the management machine and data servers. Data transmissions can be organised directly from the source storage to destination storage.



(a) Without third party transfers. (b) With third party transfers.

Figure 2.4.1: Third party transfer in grid.

Higher Level Transfer in Grid environment. However, even though the data transmission enabling the third party transfer function exhibits performance improvement, there is a problem when the user computer itself initiates the transfer. In this case, the user computer still holds information about the state of the transfer. It means that the data transmission between source storage and destination storage might fail or be terminated if the user lost connection with to the network during data transmission. More importantly, when data transfers are initiated at the client level, there is no notion of the global state of data transfers between two sites. This can lead to a number of problems [Ste06]:

- Storage elements can be overwhelmed with transfer requests;
- Clients can attempt to replicate the same file simultaneously onto the same storage element;
- Sites have little control over the use of their network resource.

Then, a service-based architecture is proposed. By providing a stateless interface, users organise and monitor data transfer through a stateless connection with a data transfer service, in this case, through web-service.

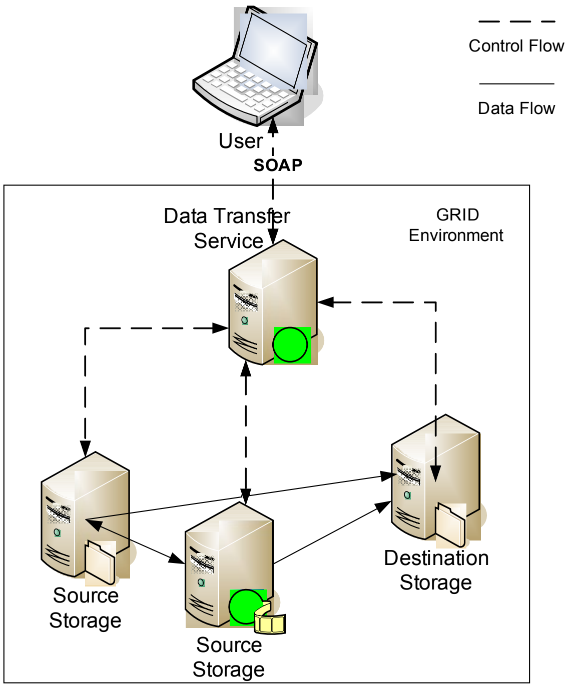


Figure 2.4.2: Architecture of Data Transfer Service

Figure 2.4.2 gives a basic architecture of the so called higher level data transfer in the grid. A web-based file transfer service is exposed to users. A user submits a transfer request, specifies quality of service requirements, monitors status, or terminates transfer through a HTTP connection to the web service. Therefore, management information is not directly passed back to the user end, but maintained within the grid environment, which brings improvements in the following aspects:

- Important information is not held in user end. Data transfer will be continued even when the user is disconnected. Robustness is improved;

- Detailed network information might not be passed to the user end. End users are allowed to interact with data transfer by using pre-agreed services. Security is improved;
- Web-service can be used by several users simultaneously, which makes it possible in coordinating data transfers to reach an optimum transmission resource distribution. Also, parallel transferring is possible where data is obtained from data replicas in different locations. This will be addressed in the remainder of this section.

This architecture has been widely used in grid middleware developments. An OGSI-compliant Reliable File Transfer (RFT) included in the Globus Toolkit [RTF] presents a stateless web service to users in clients. To submit a transfer request to RFT, users use SOAP over HTTPS, giving a list of source and destination, and specifying transfer parameters. Another example is the gLite File Transfer Services (FTS) [FTS]. It also presents a web service interface to its users. When clients submit file transfer requests, they submit a list of source URLs, destination URLs and related parameters.

2.4.2. Data Replication and Access

Data replication is a key component for large-scale distributed systems such as Content Distribution Networks (CDNs) and peer-to-peer based file sharing applications. When we want to get a kind of data from a distributed system, it may well happen that more than one storage element within the distributed system are able to provide the data needed. As in the case shown in figure 2.5.2, there are two source storage elements that can meet the requirement of the destination storage element. The advantages of having replica data in a distributed system are as follows:

- When multiple data sources exist in the distributed system, robustness of the system is significantly improved. In case any data source fails or be damaged, the data service can still operate properly thanks to replica data stored in physically different storage elements.
- With replica service, selected data can be duplicated to a storage element close to the destination. Such effort can reduce the transmission delay significantly. Evermore, in case more than one data destinations are located closely, replicate data to a nearby storage element can significant reduce long-haul data transfer, which again increase the transmission efficiency.
- When a type of data resource is supplied by multiple storage elements spread across the distributed system, its scalability improves in two aspects. First, with more data suppliers, the transmission and I/O bottleneck of source storage elements can be eliminated. Second, destination element can obtain data from multiple data sources, which provides the possibility of increasing the throughput of data transmission.

Although data replication brings benefits in terms of robustness, scalability, and performance, there are some difficulties in applying replication in the distributed system. Location of replica data is one of them. Replicate data location mechanism is needed to search and locate data. It is to identify qualified replicas and their location, respectively. Also, mechanisms dealing with the creation, registration, consistency of replica data are needed. In addition, the performance optimisation issues and security related issues are also to be considered in proposing a replication service.

In the situation where a file is stored in different sites across the Internet, how to access them becomes complex. As depicted in figure 2.4.2, in case a data transfer request is submitted through HTTP+SOAP, the data transfer service has to identify a) what data to be moved; and b) where is the source (destination) storage place. A simplified process for data transfer with replicas in grid environment is shown in figure 2.4.3. A user that requires data access sends a message to data management service. Data management service begins by querying metadata attributes with specifications of the characteristics of the desired data. Associations between key characteristic specifications and logical files are maintained and updated when needed. Then, exact logical files can be identified by applications based on their requirements. Once the logical file has been identified, the replica location service is used to locate all replica locations containing physical file instances of this logical file. Physical locations are consequently being used to decide which data resource provides best transmission performance based on network performances. Then, suitable data movement plan is decided and passed to transmission service based on certain criteria.

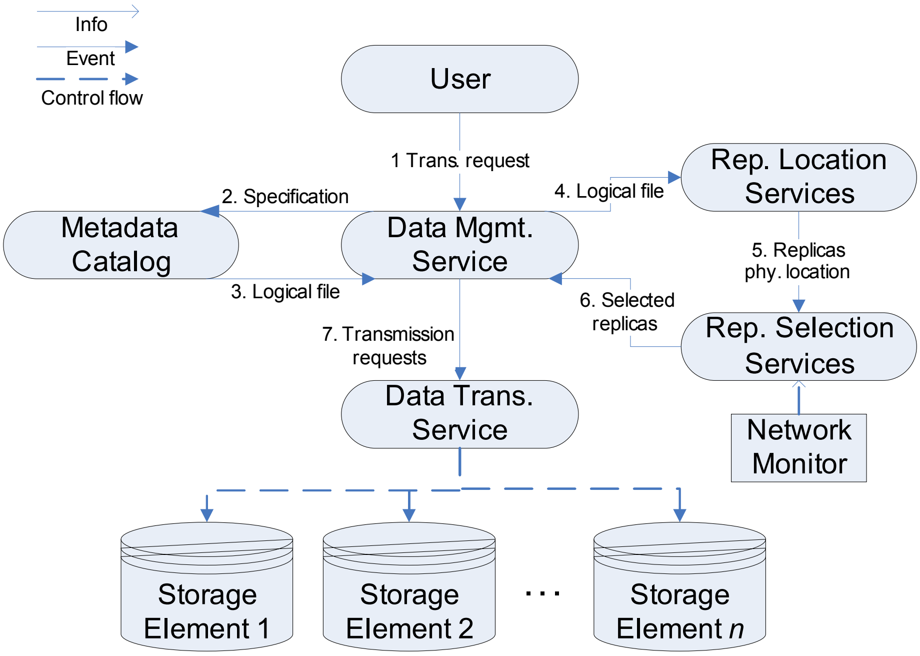


Figure 2.4.3: Data access in grid data transmission.

The design of data management system is modular with several independent services interacting via the data management service. Metadata Catalog service maintains associations between logical files and their representative characteristics. By providing representative characteristics, the service replies with its logical file identification. Replica location service serves as registries to locate where replicas exist by defining a mapping between a data logical description and the service that can provide access to the data object. With the replication location services, replicas are not constrained as "bit-wise" copies. Replica selection service locates the best replica to access based on performance and consistency considerations. In some cases, it is extended and named as optimisation service. Data transfer service is underlying transport service and provides basic mechanisms for accessing and managing the data located in storage systems. These mechanisms provide abstractions for uniformly creating, deleting, accessing and modifying file instances across storage systems. Network Monitor, e.g. Network Weather Service (NWS), is an external service and collects network operation information and passes it to replica selection service to assist their decision making.

2.4.3. Consistency Service

In a data management system with replicas, multiple replicas are spread in different physical locations. In general, consistency and synchronisation problems of replicas are not a major problem if those distributed files are set as read-only. However, a problem appears when applications might require modifying the replica data. All replicas are obliged to make an update with the same up-to-date replicas when the original file is modified. In order to allow the data management system deliver correct data, which is critical for data in any system, keeping the consistency over widely distributed replicas is an important issue. Therefore, if transmission error in replication process is put into consideration, the update synchronisation of multiple replicas of a logical file becomes the main concern of replica consistency.

To deal with the challenge, many solutions have been proposed, and some of them have been integrated to the grid system. In [Dul01], the authors discuss the concept of data consistency service for the grid environment. A strict approach guarantees that all replicas are always fully synchronised and thus fully consistent. Due to the locking overhead of keeping huge amounts of distributed data in synchronisation, complete consistency is a very impractical solution for the grid environment. Thus, if knowledge about the data and user requirements is available, one can relax this strict consistency requirement and allow certain parts of the data to be out of synchronisation for a particular amount of time. For instance, a site A in a data grid may explicitly define that newly created files at other sites B, C, and D have to be transferred to the site A within two days. This means the replica creation process can be done within a 48 hours time frame. Within this period the state of physical files can be inconsistent. Another example is that writeable replicas have to be updated and synchronised every 10 minutes. The authors have propose that a grid consistency service should support different levels of data consistency, which could be exploited by users. [Cig06]

The update scenario for data grid consistency also varies. Two protocols, named lazy-copy and aggressive copy are proposed in ref [Sun04]. Replicas are only updated as needed if someone accesses it in the lazy-copy based protocol. It can save network bandwidth resources without transferring up-to-date replicas every time when some modifications are made. However, lazy-copy protocol has to pay the penalties for access delay when inter-site updating is required. For the aggressive-copy protocol, replicas are always updated immediately when the original file is modified. In other words, full consistency for replica is guaranteed in aggressive-copy, whereas partial consistency is applied to lazy-copy. Compared with lazy-copy, access delay time can be reduced by the aggressive-copy based mechanism without suffering from long update time during each replica access. Nevertheless, full consistency with frequent replicas updates could exhaust considerable amount of network bandwidth resources. Furthermore, some updates may be invalid and inefficient because it is probable that it will never be used [Cha06].

2.4.4. Movement Planning and Bulk Data Movement Prediction

Movement planning is a key function for the replication selection service. As shown in figure 2.4.3, when the physical locations of a data file has been reported in step 5, obviously, since data can be obtained from more than one storage element and network conditions of each replica vary, a correct selection can result in performance improvements, or vice versa. The selection of replica can apply various rules. However, in a data grid without special requirements (such as data ownership or security etc.), file transfer time is always considered as the only criterion.

Obtaining accurate predictions of file transfer times between storage elements is of benefit to movement planning. Due to the shared nature of devices in the path (e.g. ethernet), their performance might vary in unpredictable manner. Since numerous devices are involved in the end-to-end path, achieving an accurate prediction can be challenging.

Some researchers use the statistics of data in data movement prediction. Those mathematical functions can be grouped as follows [Vaz02]:

Mean-Based Model. Mean-based, or averaging, techniques are a standard class of predictors that use arithmetic averaging (as an estimate of the mean value) over some portion of the measurement history to estimate future behaviour. The general formula for these techniques is the sum of the previous n values over the number of measurements. Mean-based predictors vary with the amount of history information used in their calculations and the amount of weight put on each value. For example, a total average uses the entire set of history data with each value weighted equally, but if more recent behaviour has better predictive value, then a subset of the data is used. We discuss these variations in the subsequent section.

Median-Based Model. A second class of standard predictors is based on evaluating the median of a set of values. Given an ordered list of t values, if t is odd, the median is the (t+1)/2 value; if t is even, the median is half of the t/2 value added with the (t+1)/2 value. Median-based predictors are particularly useful if the measurements contain randomly occurring asymmetric outliers that are rejected. However, they lack some of the smoothing that occurs with a mean-based method, possibly resulting in forecasts with a considerable amount of jitter.

Auto-Regression Model. A third class of common predictors is auto-regressive models. They use an Auto-regressive Integrated Moving Average (ARIMA) model technique that is constructed using the equation:

$Y_t = a + bY_{t-1}$,

where Y_t is the prediction for time, t, Y_{t-1} is the previous data occurrence and a and b are the regression coefficients that are computed based on past occurrences of Y. The standard equation includes a shock term, which is not needed in this case. This approach is most appropriate when there are at least 50 measurements and the data is measured with equally spaced time intervals. The main advantage of using an ARIMA model is that it gives a weighted average of the past values of the series thereby possibly giving a more accurate prediction. However it needs a larger data set than the previous techniques to achieve a statistically significant result, and can have a much greater computational cost.

2.4.5. Replica Placement

Replica placement optimises the performance of the data management system from another angle, which starts from where to create and maintain replicas. It is also important to the performance of the data management system. Place replica close to the data consumer can greatly reduce the usage of bandwidth resource in transmission and eliminate transmission delays. Given a certain amount of space and location to construct replicas, an optimal replica placement problem occurs.

Optimal replica placement problem has been studied extensively in the literature. The same problem has different names in different research areas. For example, it is refereed to as p-median problem in operations research, or database location problem on Internet and file allocation problem in computer science. Wolfson and Milo proved that replica placement problem is NP-Complete for general graphs when read and update cost are simultaneously considered. They also provide optimal solutions for special topologies, including complete graph, tree, and ring. Tu and Xu study the secure data placement problem in the same model and provide a heuristic algorithm for general graphs. Krick et al. consider read, update and storage cost simultaneously in general graph, and provide a polynomial time approximation algorithm that has a constant competitive ratio. They also provide an optimal solution for tree topology in the same paper. Kalpakis, Dasgupta and Wolfson consider read, update and storage cost under tree topology. Their algorithm could cope with the situations even when servers have capacity limits. They describe an O (n^3p^2) dynamic programming algorithm for p replicas placed in n uncapacitated servers, and an O ($n^3p^{2\wedge}{}_{max}{}^2$) algorithm for capacitated servers, where $^\wedge{}_{max}$ denotes the maximum capacity among all servers. Unger and Cidon provide a more efficient algorithm to find the optimal placement under the same model, with only O (n^2) time, where n is the number of servers [Wan06].

2.4.6. Example of Grid Data Management System

The architecture shown in figure 2.4.3 and functions elaborated above are adapted to many projects with slight amendments. The European DataGrid project heavily involves in the replication management and implements the data replication service as follows. The main components of the EU DataGrid data are as shown in figure 2.4.4.

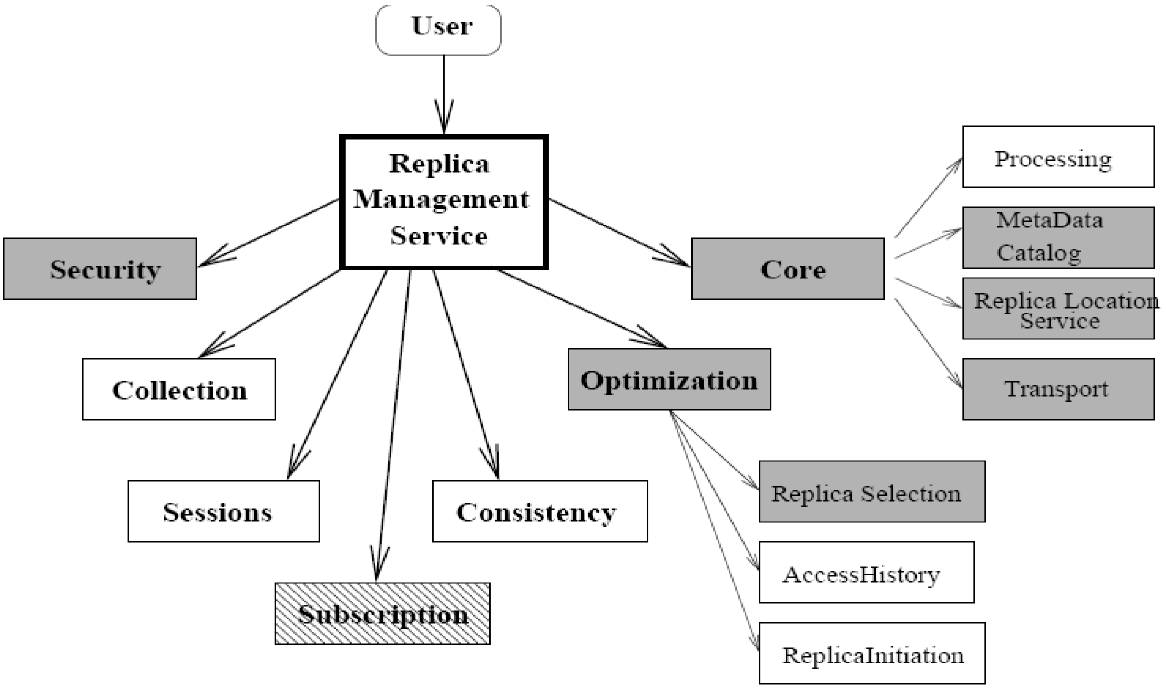


Figure 2.4.4: Replica Management Service's main design, components in EU DataGrid.

As reported in [Bos03], the Replica Management Service (RMS) acts as a logical single entry point to the system, and interacts with the other components of the system. The core component provides the main function of RMS, and interacts with third party modules. The optimization component is to improve the performance of the RMS by access appropriate replicas. This structure is implemented using the Java J2EE framework and SOAP remote procedure call. The client interface is provided via Java API.

2.5. Workflow Description: The "Business Process Execution Language for Web Services" (BPEL4WS) Standard

Scientific experiments, as well as other kinds of applications with multiple actors and logical activities, require the application of the workflow concept. A workflow can be defined as a construct which denotes the way information is being exchanged between the different actors within the context of an application, and indicates their temporal or other kinds of dependencies. Essentially, a workflow shows how tasks are structured within a process, what activity follows another and what activity retrieves input from another. In the grid context, workflow languages are necessary as the components of applications are loosely coupled and are typically (re-)composed to create new applications from the same structural blocks.

In the last decades, there have been numerous efforts to define and implement workflow languages, which can describe adequately a workflow and its actors. Some of them are geared specifically towards web services, while others are more generic. We are listing the more important such efforts below, focusing on the ones which arestandardized or :

- **The XML Process Definition Language (XPDL)** is a format standardized by the Workflow Management Coalition (WfMC) to interchange Business Process definitions between different workflow products like modeling tools and workflow engines. XPDL defines a XML schema for specifying the declarative part of workflow. XPDL is designed to exchange the process design, both the graphics and the semantics of a workflow business process. XPDL contains elements to hold the X and Y position of the activity nodes as well as the coordinates of points along the lines that link those nodes. This distinguishes XPDL from BPEL which is also a process definition format, but BPEL focuses exclusively on the executable aspects of the process. BPEL does not contain elements to represent thegraphical aspects ofa process diagram[XPDL].
- **Yet Another Workflow Language (YAWL)** is a workflow language based on the Workflow patterns. The language is supported by a software system that includes an execution engine and a graphical editor. The language and its supporting system were originally developed by researchers at Eindhoven University of Technology and the Queensland University of Technology. Subsequently, several organizations such as InterContinental Hotels Group, first:telecom and ATOS Worldline have joined this initiative and the YAWL system is now available as an Open source software under the LGPL license [YAWL].
- **Business Process Execution Language (BPEL)** for web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment — even across multiple organizations — using a combination of web services. Written by developers from BEA Systems, IBM, and Microsoft, BPEL combines and replaces IBM's WebServices Flow Language (WSFL) and Microsoft's XLANG specification. (BPEL is also sometimes identified as BPELWS or BPEL4WS.). It aims to enable programming in the large. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes [BPEL].
- **Business Process Modeling Language (BPML)** is a meta-language for the modeling of business processes, just as XML is a meta-language for the modeling of business data. BPML was a proposed language, but now the BPMI has dropped support for this in favor of BPEL4WS [BPML].
- **Web Services Flow Language (WSFL)** is an XML language proposed by IBM to describe the composition of web services. WSFL has been superseded by BPEL [WSFL].
- **Wf-XML** is a business process management standard developed by the Workflow Management Coalition (WfMC). Wf-XML is designed and implementedas an extension to the OASIS Asynchronous Service Access Protocol (ASAP). ASAP provides a standardized way that a program can start and monitor a program that might take a long time to complete. It provides the capability to monitor the running service, and be informed of changes in its status. Wf-XML extends this by providing additional standard web service operations that allow sending and retrieving the "program" or definition of the service which is provided. A process engine has this behavior of providing a service that lasts a long time, and also being programmable by being able to install process definitions [WfXML].

BPEL has become the de-facto standard flow language to support the notion of Service Oriented Architecture where multiple web services are combined in a predefined scenario in order to build higher level services. More and more implementations of the language appear, due to its generic constructs which allow for short and long running processes, as well as the fact that it can be expanded with abstract definitions attached to BPEL documents.

2.5.1. Service Oriented Architecture

Programming an application is becoming more and more complex over the years creating the need for new programming patterns in order to simplify the process of writing software. For example, object oriented programming appeared as a programming pattern that provides reusable code between applications. However, in today's environment were distributed software, application integration, varying platforms, varying protocols, various devices, the Internet etc come into account, there are issues that need to be addressed in application development.

SOA is a new programming pattern/architecture for building applications. It is more than a particular set of technologies, such as web services; it transcends them, and, in the best-case scenario, is totally independent of them. Within a business environment, a pure architectural definition of a SOA might be something like "an application architecture within, which all functions are defined as independent services with well defined invokable interfaces which can be called in defined sequences to form business processes". The key components of a SOA are:

- All functions are defined as services. This includes purely business functions, business transactions composed of lower-level functions, and system service functions. This brings up the question of granularity, which will be addressed later.
- All services are independent. They operate as "black boxes"; external components neither know nor care how they perform their function, merely that they return the expected result.
- In the most general sense, the interfaces are invokable and more specifically are dynamically invokable; that is, on an architectural level, it is irrelevant whether they are local (within the system) or remote (external to the immediate system), what interconnect scheme or protocol is used to effect the invocation, or what infrastructure components are required to make the connection. The service may be within the same application, or in a different address space within an asymmetric multiprocessor, on a completely different system within the corporate Intranet, or within an application in a partner's system used in a B2B configuration.
- Another factor is the dynamic discovery of the services.

In the world of web services the above are addressed by SOAP for the interface, WSDL for dynamic invocation, and UDDI for discovery. web services also have the feature of being independent of each other. That is why web services are an excellent candidate for a Service Oriented Architecture.

In all this, the interface is the key, and is the focus of the calling application. It defines the required parameters and the nature of the result; thus, it defines the nature of the service, not the technology used to implement it. It is the system's responsibility to effect and manage the invocation of the service, not the calling application. This allows two critical characteristics to be realized: first, that the services are truly independent, and second, that they can be managed. Management includes many functions, including:

- **security:** authorization of the request, encryption and decryption as required, validation, etc.
- **deployment:** allowing the service to be redeployed (moved) around the network for performance, redundancy for availability, or other reasons
- **logging:** for auditing, metering, etc.

- **dynamic rerouting:** for fail over or load balancing
- **maintenance:** management of new versions of the service

By defining SOA we have seen that the general idea is to build new services by using existing ones. To do this we need to link the services together into a business process by defining the workflow of that process. The Business Process Execution Language for Web Services (BPEL4WS) represents the emerging standard for describing business processes in the world of web services. BPEL4WS combines the best of both WSFL (support for graph oriented processes) and XLANG (structural constructs for processes) into one cohesive package that supports the implementation of any kind of business process in a very natural manner. In addition to being an implementation language, BPEL4WS can be used to describe the interfaces of business processes as well — using the notion of abstract processes.

2.5.2. BPEL4WS Concepts

BPEL4WS supports two distinct usage scenarios:

1. Implementing executable business processes.
2. Describing non-executable abstract processes.

As an executable process implementation language, the role of BPEL4WS is to define a new web service by composing a set of existing services. Thus, BPEL4WS is basically a language to implement such a composition. The interface of the composite service is described as a collection of WSDL portTypes, just like any other web service. The composition (called the process) indicates how the service interface fits into the overall execution of the composition. Illustration 1 outlines this outer view of a BPEL4WS process.

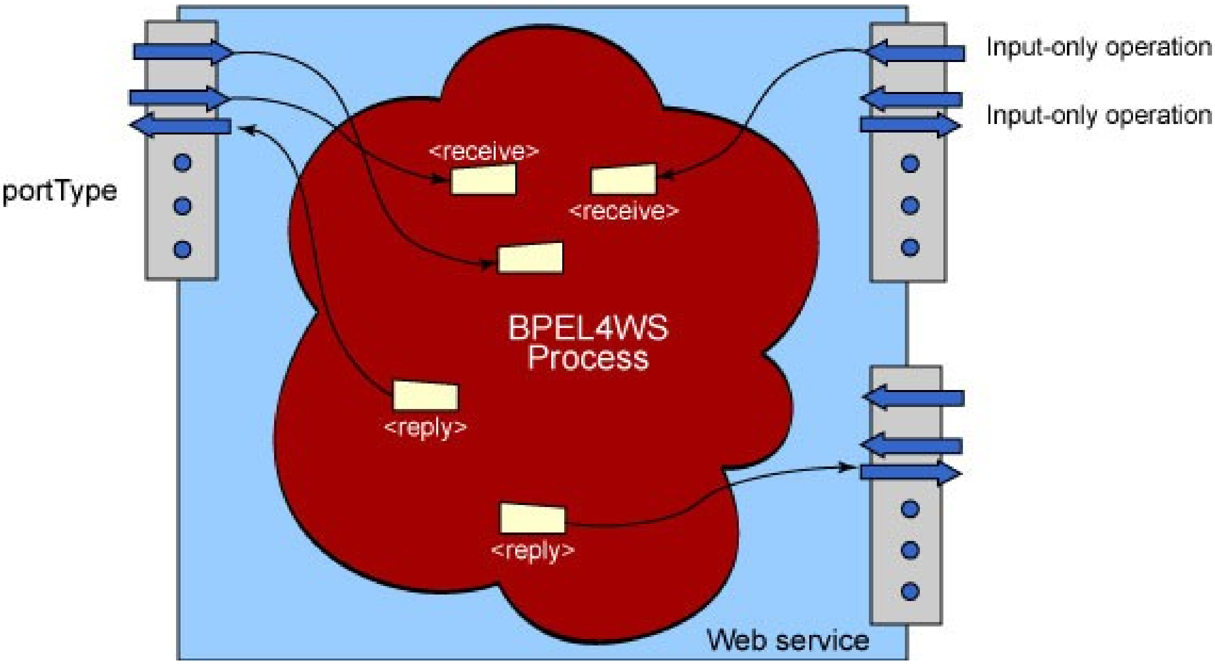


Figure 2.5.1: Input-output operation in a BPEL4WS process.

The composition primitives found in BPEL4WS come primarily from many years of experience in workflow and business process integration, hence its positioning as a business process composition language.

2.5.3. Implementing the service

What's in the cloud in the figure above? Unlike a traditional programming language implementation of a WSDL service, each operation of each portType does not map to a separate piece of logic in BPEL4WS. Instead, the entire type of the service (that is, the set of `portTypes` of the service) is implemented by one single BPEL4WS process. Thus, specific "entry-points" corresponding to external users invoking the operations of the interface are indicated within the BPEL4WS description. These entry points either consume WSDL operations' incoming messages from input-only or input-output operations. In the latter case, the process must also indicate where the output message is generated. BPEL4WS only uses and supports input-only and input-output (request-response) operations of WSDL; output-only (notification) and output-input (solicit-response) operations are not required nor supported.

The BPEL4WS process itself is basically a flow-chart like expression of an algorithm. Each step in the process is called an activity. There are a collection of primitive activities: invoking an operation on some web service (`<invoke>`), waiting for a message to operation of the service's interface to be invoked by someone externally (`<receive>`), generating the response of an input/output operation (`<reply>`), waiting for some time (`<wait>`), copying data from one place to another (`<assign>`), indicating that something went wrong (`<throw>`), terminating the entire service instance (`<terminate>`), or doing nothing (`<empty>`).

These primitive activities can combined into more complex algorithms using any of the structure activities provided in the language. These are the ability to define an ordered sequence of steps (`<sequence>`), the ability to have branching using the now common "case-statement" approach (`<switch>`), the ability to define a loop (`<while>`), the ability to execute one of several alternative paths (`<pick>`), and finally the ability to indicate that a collection of steps should be executed in parallel (`<flow>`). Within activities executing in parallel, one can indicate execution order constraints by using the links.

BPEL4WS allows you to recursively combine the structured activities to express arbitrarily complex algorithms that represent the implementation of the service.

2.5.4. Interacting with Others: Partners

As a language for composing together a set of services into a new service, BPEL4WS processes mainly consist of making invocations to other services and/or receiving invocations from clients (the users of the service in illustration 2.5.1). The prior is done using the `<invoke>` activity and the latter using the `<receive>` and `<reply>` activities. BPEL4WS calls these other services that interact with a process partner. Thus, a partner is either a service the process invokes (invoked partners) as an integral part of its algorithm, or those that invoke the process (client partners).

The first kind of partners is obvious — the process must clearly invoke other services to do things. The `<invoke>` activity indicates the partner to invoke and what operation of which of the partner's `portTypes` to invoke on that partner. However, invoked partners may end up being clients as well — it may be the case that the process invokes an operation on the partner to request some service. Later on, the partner may invoke an operation on the process to provide the desired data.

The reason for treating clients of the process as partners may not be so obvious. There are actually two reasons for it: the first is that sometimes the process may need to invoke operations on one of its client partners. This is primarily how asynchronous interaction is supported: the client invokes an operation on the process to request some service. Upon completion, the process invokes an operation on the client partner. At that point, there is no distinction between a client partner and an invoked partner.

The second reason is that the service offered by the process may be used (as a whole or in parts) by more than one client. In addition, the process may wish to distinguish between these different users of the service. For example, a process representing a loan servicing system offers a single web service, but only parts of it are accessible to the customer applying for the loan, other parts for the customer service representative and finally the entire service to the loan underwriters. Depending on whether an operation is invoked by the customer or by the underwriter, the returned behavior may be quite different. Furthermore, the approach of using partners to model clients allows the process to indicate that certain operations may only be invoked by certain clients.

So, partners are one of the following:

- only services that the process invokes,
- only services that invoke the process,
- or services that the process invokes and invoke the process (where either may occur first).

The first two are straightforward invoked partners and client partners, respectively. Consider the relationship between the process and the service for the third case when the process invokes the service first. That means that the service provides (or publishes) a portType (PT1) and the process invokes an operation of that portType. Also, the process must provide a portType (PT2) that the service invokes an operation out of. Thus, from the point of view of the process, the process requires the portType PT1 from the service and provides the portType PT2 to the service. Looking at the same relationship from the point of view of the service leads to opposite statement: The service offers the portType PT1 to the process and requires the portType PT2 from the process. The situation is the same whether the process invokes the service first or vice-versa.

2.5.5. Partner Link Types

Modeling the third kind of services is what gives rise to partner link types. Instead of defining the relationship between the service and the process from the point of view of one of these participants, a partner link type represents a third party declaration of a relationship between two (or more, potentially) services. A partner link type defines a collection of roles, where each role indicates a portType. The idea is that when two services interact with each other, the partner link type is a declaration ofhow they interact — essentially what each party offers.

Basically, a partner is defined by giving it a name and then indicating the name of a partner link type and identifying the role that the process will play from that partner link type and the role that the partner will play. In the pure invoked partner and pure client partner cases, the partner link type will have just one role in it and, hence, only one is indicated at partner definition time. The partner name is then used in `<receive>`, `<reply>` and `<invoke>` activities to indicate the desired partner.

2.5.6. Endpoint References

How does a partner work at runtime? In order for it to work at runtime, the partner must resolve to an actual web service. Thus, a partner is really eventually just a typed endpoint reference, where the typing comes from the partner link type and the roles. The BPEL4WS process itself does not indicate how a partner is bound to a specific endpoint; that is considered a deployment time or runtime binding step that must be supported by the BPEL4WS implementation.

2.5.7. Dealing with Problems

Developers need ways to handle and recover from errors in business processes. BPEL4WS has exceptions (faults) built into the language via the `<throw>` and `<catch>` constructs. The fault concept on BPEL4WS is directly related to the fault concept on WSDL and in fact builds on it.

In addition, BPEL4WS supports the notion of compensation, which is a technique for allowing the process designer to implement compensating actions for certain irreversible actions. For example, in a travel reservation process, once a reservation has been confirmed, one must perform some explicit operations to cancel that reservation. Those actions are called "compensating actions" for the original action.

Fault handling and compensating is supported recursively in BPEL4WS by introducing the notion of a scope, which is essentially the unit of fault handling and/or compensation.

2.5.8. Lifecycle of Services

What about the lifecycle of these services? Web services implemented as BPEL4WS processes have an instanced life cycle model. That is, a client of these services always interacts with a specific instance of the service (process). So how does the client create an instance of the service?

Unlike traditional distributed object systems, in BPEL4WS instances are not created via a factory pattern. Instead, instances in BPEL4WS are created implicitly when messages arrive for the service. That is, instances are identified not by an explicit "instance ID" concept, but by some key fields within data messages. For example, if the process represents an order fulfillment system, the invoice number could be the "key" field to identify the specific instance involved with the interaction. Thus, if a matching instance isn't available when a message arrives at a "startable" point in the process, a new instance is automatically created and associated with the key data found in the message. Messages can only be accepted at non-startable points in a process after a suitable instance has been located; that is, in these cases the messages are in fact always delivered to specific instances. In BPEL4WS, the process of finding a suitable instance or creating one if necessary is called messagecorrelation.

2.5.9. Example

The following example has been retrieved from [FlowsWithBPEL4WS] to illustrate the basic parts of a BPEL document that describe a business process.

The PayFlow business flow example is comprised of a client initiating a request to a BPEL process and ending with the process calling back the client with the result of the payment request (receipt). The process includes use of `<receive>` and `<invoke>` activities for interacting with the outside world, which includes the client (request) and a partner (payment processor service). XML Variables are used for holding messages exchanged between the process and the partners. To make this example interesting, the payment processor service is asynchronous and can take anywhere from several minutes to several days before the service calls back the process. Another interesting element demonstrated by this example is the handling of exceptions and managing of timeouts. These constructs are instrumental to enable a BPEL process to deliver reliable business flows.

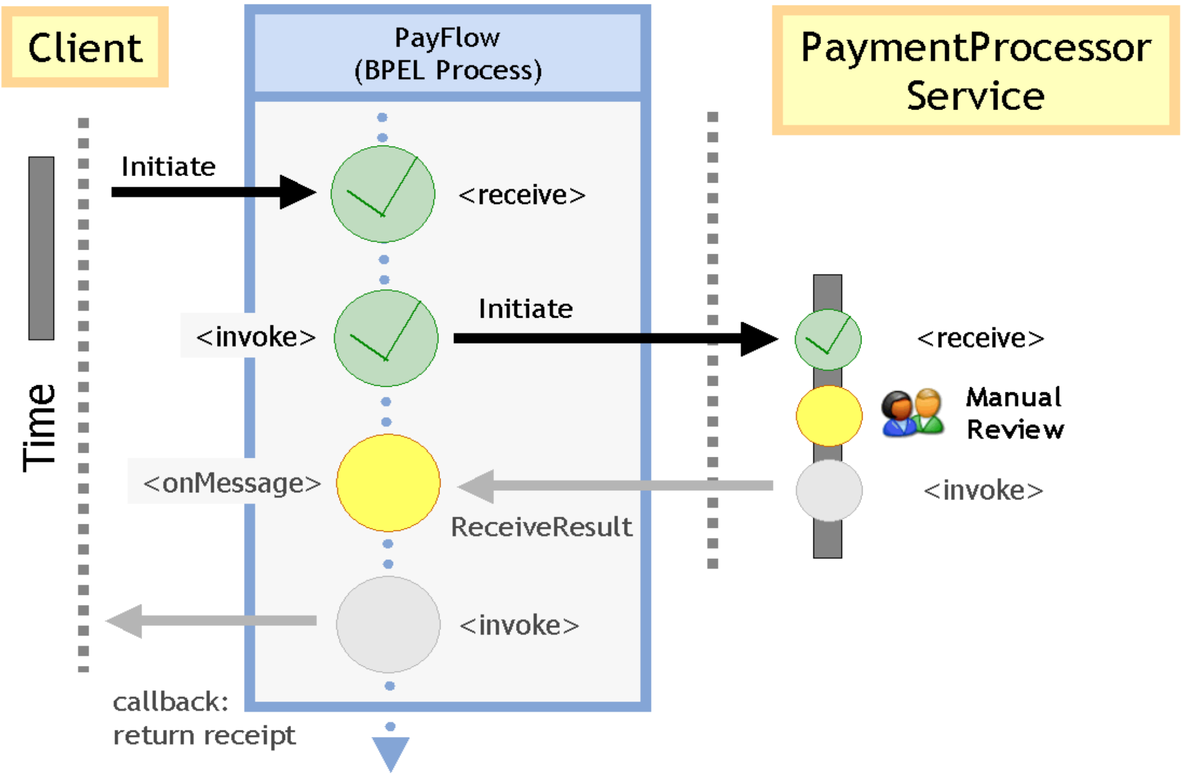


Figure 2.5.2: BPEL example.

The client initiating the PayFlow process specifies a transfer amount in dollars. The PayFlow process receives the transfer request and leverages an asynchronous payment processor to carry out the fund transfer. The payment processor is asynchronous because it requires a manual review prior to returning the result to the process (note though, that this fact is opaque to the PayFlow process that uses the service; PayFlow just knows that PaymentProcessor is an asynchronous and potentially long-running service).

The payment processor service, as implemented, will return a normal result if the amount is less than $500. Otherwise, it will return a "transfer refused" fault, which will be handled as an exception by the PayFlow BPEL process. PayFlow also handles an "insufficient funds" fault, which may be thrown by the service. In addition, the PayFlow process specifies a timeout period and terminates if the payment processor takes more than 2 days (the timeout limit) to return a digital receipt indicating a successful fund transfer

2.5.10. Conclusions

Adopting a Service Oriented Architecture for building software has many advantages in terms of distributed systems, scalability, reusability etc., as seen in previous chapters. Web service are an excellent candidate technology for supporting a SOA especially with the support of workflow language for describing business flows between the services. BPEL is currently the dominant standard for describing workflows and many tools are appearing that support this specification.

2.5.11. References

[NewToSOAandWS] IBM, "New to SOA and Web Services", http://www-106.ibm.com/developerworks/webservices/newto/

[WhatIsSOA] Hao He, "What is ServiceOriented Architecture", http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[MigratingToSOA] Kishore Channabasavaiah, Kerrie Holley, Edward M. Tuggle, Jr., "Migrating to Service Oriented Architecture", ftp://www6.software.ibm.com/software/developer/library/ws-migratesoa.pdf

[SOAExplained] Sayed Hashimi, "Service-Oriented Architecture Explained", http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html

[BPEL4WS] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, "Business Process Execution Language for Web Services Version 1.1", http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[LearningBPEL] S. Weerawarana, F. Curbera, "Business Process with BPEL4WS: Learning BPEL4WS (Part 1)", http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/ as well as Rania Khalaf, "Business Process with BPEL4WS: Learning BPEL4WS (Part 2)", http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol2/

[FlowsWithBPEL4WS] Doron Sherman, "Business Flows with BPEL4WS", http://www.sys-con.com/WebServices/articleprint.cfm?id=572

[XPDL] http://en.wikipedia.org/wiki/XPDL

[YAWL] http://en.wikipedia.org/wiki/YAWL

[BPEL] http://en.wikipedia.org/wiki/BPEL

[BPML] http://en.wikipedia.org/wiki/BPML

[WSFL] http://en.wikipedia.org/wiki/Web_Services_Flow_Language

[WfXML] http://en.wikipedia.org/wiki/Wf-XML

2.6. Interchangeable Virtual Instrument Specification

IVI is a consortium founded to promote specifications for programming test instruments that simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance.

In today's world, two factors hinder efficient test system setup and support: 1) the high cost of developing and maintaining test system software and, 2) rapidly evolving technology. The IVI Foundation addresses these needs through new driver technology:

- IVI drivers define a new level of quality, completeness, usability, and functionality that reduces the cost of test system development and ownership.
- IVI drivers simplify upgrading or replacing components in complex test systems intended to be used over a long period of time;

2.6.1. Goals of the IVI Foundation

Hardware Interchangeability: Simplify the task of replacing an instrument from a system with a similar instrument. Preserve test software when instruments become obsolete. Simplify test code reuse from design validation to production test.

Quality: Improve driver quality. Establish guidelines for driver testing and verification.

Software Interoperability: Provide an architectural framework that allows users to easily integrate software from multiple vendors. Provide standard access to driver capabilities such as range checking and state caching. Simulate instruments and develop test system software when instruments are not physically available. Provide consistent instrument control in popular programming environments.

2.6.2. IVI Class Specifications

To enable interchangeability, the foundation creates IVI class specifications that define the base class capabilities and class extension capabilities for some of the most popular instrument classes. There are currently eight instrument classes defined:

1. DC power supply
2. digital multimeter (DMM)
3. function generator & arb

4. oscilloscope
5. power meter
6. RF signal generator
7. spectrum analyzer
8. switch

Future work includes defining additional class specifications and extending the scope of current specifications to cover more instrument functionality.

2.6.3. IVI Driver Architecture

IVI drivers provide many inherent capabilities that go beyond those of traditional instrument drivers. The IVI specifications have been developed to enable drivers with a consistent and high standard of quality, usability, and completeness. The specifications define advanced features such as instrument simulation, state caching, automatic range checking, and multithread safety. In addition, IVI Foundation members have cooperated to provide common software components that ensure multi-vendor system compatibility. IVI custom specific drivers support only these inherent capabilities and instrument specific capabilities that are not standardized upon by the foundation and that are unique to a particular instrument.

In addition to these inherent capabilities, IVI drivers can comply with an instrument class specification to support the foundation's goal of instrument interchangeability. These drivers include:

- Base class capabilities common to most instruments in a class (e.g., edge-triggered acquisition on a scope);
- Class extension capabilities that represent more specialized features of an instrument class (e.g., TV or width trigger on a scope).

IVI class-compliant specific drivers contain inherent capabilities, base class capabilities, as well as class extension capabilities that the instrument supports. To achieve interchangeability, users program to an IVI class interface available through an IVI class-compliant specific driver or a separate IVI class driver.

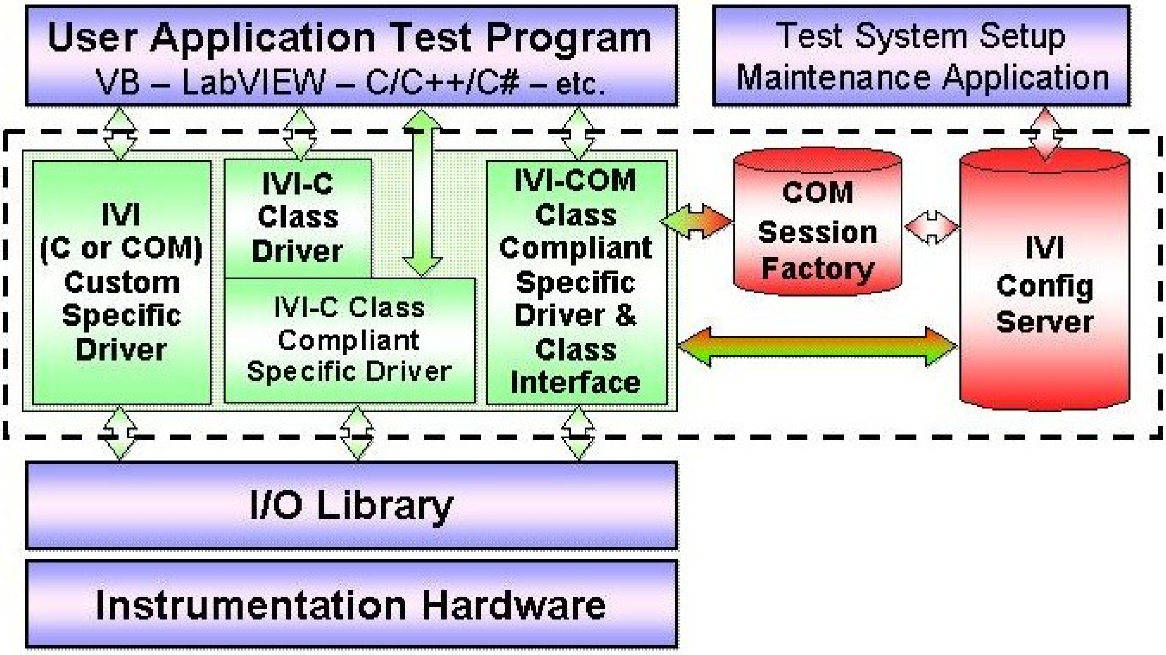


Figure 2.6.1: Typical use of IVI framework.

To support all popular programming languages and development environments, IVI drivers provide either a C or a COM API. Driver developers may provide both interfaces as well as interfaces optimized for specific development environments.

All IVI drivers communicate to the instrumentation hardware through an I/O Library. The VISA library is used for the GPIB and VXI buses, while other buses can either utilize VISA or another library.

Future work includes Measurement and Stimulus Subsystems and Signal Interfaces to provide interchangeability in more specialized and complex situations where the current architecture is insufficient. Additional efforts are underway to pursue the use of IVI drivers in Microsoft .NET environments.

IVI Conformance. IVI drivers that conform to and are documented according to the IVI specifications may display the IVI conformance logo for easy identification. Instruments or software applications may display the standard IVI logo to indicate conformance with the IVI specifications. A product that does not display an IVI logo is non-conformant.

Figure 2.6.2: IVI logo.

Driver API. To support all popular programming languages and development environments, IVI drivers provide either a IVI-C or an IVI-COM API. Driver developers may provide both interfaces, as well as wrapper interfaces optimized for specific development environments.

Instrument I/O. All IVI drivers communicate to the instrumentation hardware through an I/O Library. The VISA library is used for the GPIB and VXI buses, while other buses can either utilize VISA or another library.

2.6.4. IVI Driver Specifications

The IVI Foundation currently standardizes on two interface technologies: COM and ANSI-C. IVI drivers conform to the IVI-COM architecture, the IVI-C architecture, or both. IVI driver suppliers choose which architectures to support based on the needs of their customers. As computer and software technology evolves, other interface technologies may become popular within the instrument control community. As this change occurs, new interfaces may be defined to incorporate new capabilities.

Revision Notes. The first draft of the IVI Driver Architecture Specification was released in July 2000. The latest revision number is 1.6, released in January 11, 2007.

2.6.5. IVI-COM Driver Architecture

COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX® Controls.

Target Operating Systems. IVI-COM drivers work on the following Microsoft operating systems: Windows 2000, Windows XP, Windows Vista 32, and Windows Vista 64 (32-bit applications only). In principle, IVI-COM drivers can be developed for other operating systems on which COM is available, including Sun Solaris and Linux.

For the minimum service pack level required to use the IVI shared components on each operating system, refer to the download page on the IVI Foundation web site, www.ivifoundation.org.

Target Languages and Application Development Environments. IVI-COM drivers work in Microsoft Visual C++, Microsoft Visual Basic, Microsoft Visual Basic for Applications, National Instruments LabVIEW, and National Instruments LabWindows/CVI. In principle, IVI-COM drivers can work in other development environments in which COM is supported, including Borland C/C++.

2.6.6. IVI-C Driver Architecture

Target Operating Systems. IVI-C drivers work on the following Microsoft operating systems: Windows 2000, Windows XP, Windows Vista 32, and Windows Vista 64 (32-bit applications only). IVI-C drivers can also work on any other operating system if the following conditions are met:

- A compiled version of the IVI-C driver is available, or source code is available and an ANSI-C compiler is available for that operating system.
- The C shared components are compiled and available for that operating system.
- An I/O library that the IVI-C driver uses is available for that operating system.
- Any other support libraries that the driver uses areavailable for that operating system.

To enable use on other operating systems, IVI-C drivers should avoid making operating system specific calls.

For the minimum service pack level required to use the IVI shared components on each operating system, refer to the download page on the IVI Foundation web site, www.ivifoundation.org.

For an example of driver developments on other operating systems look at "Porting IVI-C Specific Drivers and Applications to Linux" http://zone.ni.com/devzone/cda/tut/p/id/3809

T**arget Languages and Application Development Environments.** IVI-C drivers work in Microsoft Visual C/C++, National Instruments LabVIEW, and National Instruments LabWindows/CVI. In principle, IVI-C drivers can work in other ADEs that allow calls to 32-bit dynamic link libraries, such as Borland C/C++, MathWorks MATLAB, and Agilent VEE.

3. Middleware

3.1. Globus Toolkit

Globus Toolkit, currently at version 4, is an open source toolkit for building computing grids provided by the Globus Alliance.

It is the most widely diffused enabling technology for the "grid". It includes software services and libraries for resource monitoring, discovery, and management, plus security and file management.

It is packaged as a set of components that can be used either independently or together to develop applications. Its core services, interfaces and protocols allow users to access remote resources preserving local control over who canuse resources and when.

The open source Globus Toolkit is a fundamental enabling technology for the "grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. In addition to being a central part of science and engineering projects that total nearly a half-billion dollars internationally, the Globus Toolkit is a substrate on which leading IT companies are building significant commercial grid products.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

The Globus Toolkit has grown through an open-source strategy similar to the Linux operating system's, and distinct from proprietary attempts at resource-sharing software. This encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

Essential background is contained in the papers "Anatomy of the Grid" by Foster, Kesselman and Tuecke and "Physiology of the Grid" by Foster, Kesselman, Nick and Tuecke.

The Globus Toolkit is an implementation of the following standards:

- Open Grid Services Architecture (OGSA)
- Open Grid Services Infrastructure (OGSI) — originally intended to form the basic "plumbing" layer for OGSA, but has been superseded by WSRF and WS-Management.
- Web Services Resource Framework (WSRF)
- Job Submission Description Language (JSDL)
- Distributed Resource Management Application API (DRMAA)
- WS-Management
- WS-BaseNotification
- SOAP

- WSDL
- Grid Security Infrastructure (GSI)

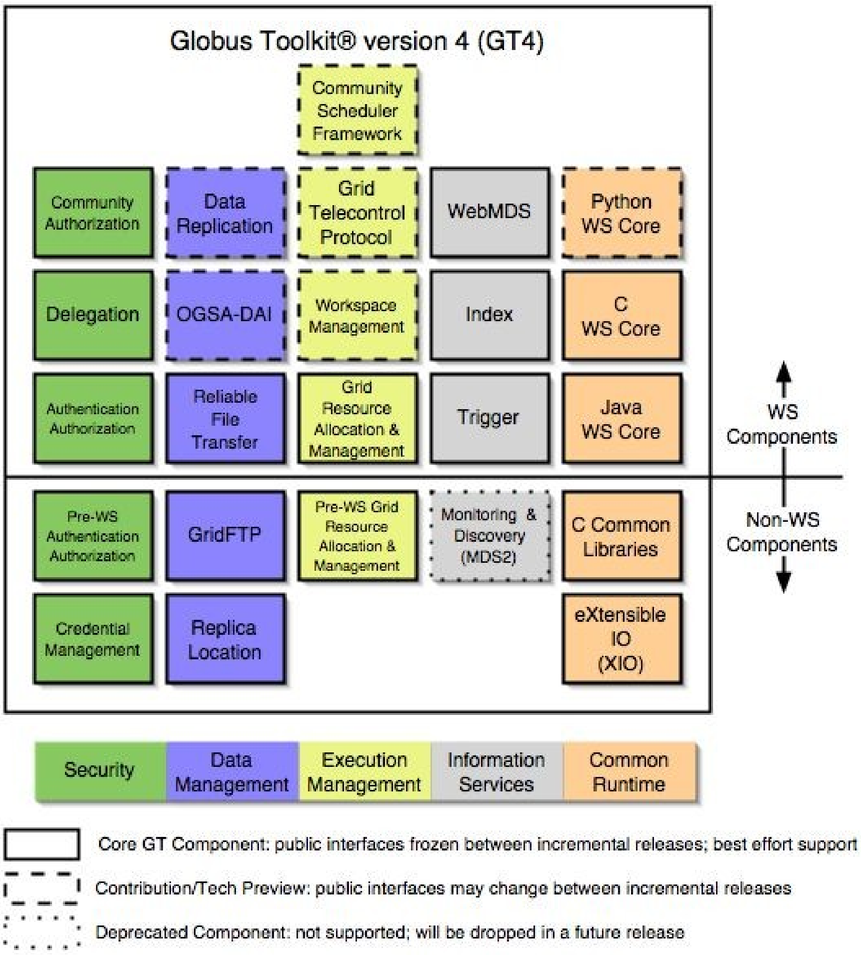

Globus Toolkit® version 4 (GT4)
Community Scheduler Framework
Community Authorization
Data Replication
Grid Telecontrol Protocol
WebMDS
Python WS Core
Delegation
OGSA-DAI
Workspace Management
Index
C WS Core
Authentication Authorization
Reliable File Transfer
Grid Resource Allocation & Management
Trigger
Java WS Core
WS Components
Non-WS Components
Pre-WS Authentication Authorization
GridFTP
Pre-WS Grid Resource Allocation & Management
Monitoring & Discovery (MDS2)
C Common Libraries
Credential Management
Replica Location
eXtensible IO (XIO)
Security
Data Management
Execution Management
Information Services
Common Runtime
Core GT Component: public interfaces frozen between incremental releases; best effort support
Contribution/Tech Preview: public interfaces may change between incremental releases
Deprecated Component: not supported; will be dropped in a future release

Figure 3.1.1: Globus Toolkit components.

3.1.1. Acclaim for the Globus Toolkit

From version 1.0 in 1998 to the 2.0 release in 2002 and now the latest 4.0 version based on new open-standard grid services, the Globus Toolkit has evolved rapidly into what The New York Times called "the de facto standard" for grid computing. In 2002 the project earned a prestigious R&D 100 award, given by R&D Magazine in a ceremony where the Globus Toolkit was named "Most Promising New Technology" among the year's top 100 innovations. Other honors include project leaders Ian Foster of Argonne National Laboratory and the University of Chicago, Carl Kesselman of the University of Southern California's Information Sciences Institute (ISI), and Steve Tuecke of Argonne being named among 2003's top ten innovators by InfoWorld magazine, and a similar honor from MIT Technology Review, which named Globus Toolkit-based grid computing one of "Ten Technologies That Will Change the World." The Globus Toolkit also won the 2003 Federal Laboratory Consortium award for excellence in technology transfer, in recognition of its widespread adoption by industry.

3.1.2. Genesis of the Globus Project

In late 1994 Rick Stevens, director of the mathematics and computer science division at Argonne National Laboratory, and Tom DeFanti, director of the Electronic Visualization Laboratory at the University of Illinois at Chicago, proposed establishing temporary links among 11 high-speed research networks to create a national grid (the "I-WAY") for two weeks before and during the Supercomputing '95 conference. A small team led by Ian Foster at Argonne created new protocols that allowed I-WAY users to run applications on computers across the country. This successful experiment led to funding from the Defense Advanced Research Projects Agency (DARPA, and 1997 saw the first version of the Globus Toolkit, which was soon deployed across 80 sites worldwide. The U.S. Department of Energy (DOE) pioneered the application of grids to science research, the National Science Foundation (NSF) funded creation of the National Technology Grid to connect university scientists with high-end computers, and NASA started similar work on its Information Power Grid.

3.1.3. Widespread Adoption of the Globus Toolkit

Grids first emerged in the use of supercomputers, as scientists and engineers across the U.S. sought access to scarce high-performance computing resources that were concentrated at a few sites. Begun in 1996, the Globus Project was initially based at Argonne, ISI, and the University of Chicago (U of C). What is now called the Globus Alliance has expanded to include the University of Edinburgh, the Royal Institute of Technology in Sweden, the National Center for Supercomputing Applications, and Univa Corporation. Project participants conduct fundamental research and development related to the grid. Sponsors include federal agencies such as DOE, NSF, DARPA, and NASA, along with commercial partners such as IBM and Microsoft.

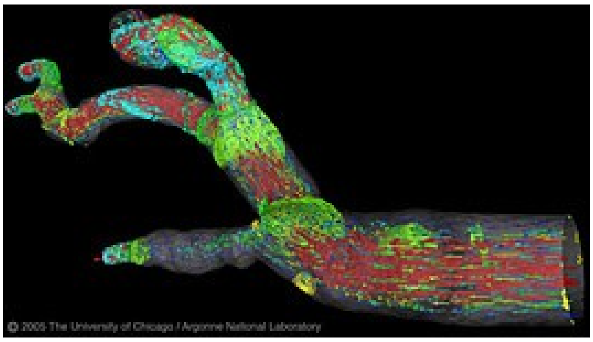
The project has spurred a revolution in the way science is conducted. High-energy physicists designing the Large Hadron Collider at CERN are developing Globus-based technologies through the European Data Grid, and the U.S. efforts like the Grid Physics Network (GriPhyN) and Particle Physics Data Grid. Other large-scale e-science projects relying on the Globus Toolkit include the Network for Earthquake Engineering and Simulation (NEES), FusionGrid, the Earth System Grid (ESG),the NSF Middleware Initiative and its Grids Center, and the National Virtual Observatory. In addition, many universities have deployed campus grids, and deployments in industry are growing rapidly.

Much as the World Wide Web brought Internet computing onto the average user's desktop, the Globus Toolkit is helping to bridge the gap for commercial applications of grid computing. Since 2000, companies like Avaki, DataSynapse, Entropia, Fujitsu, Hewlett-Packard, IBM, NEC, Oracle, Platform, Sun and United Devices have pursued grid strategies based on the Globus Toolkit. This widespread industry adoption has brought a new set of objectives, with the cardinal purpose being to preserve the open-source, non-profit community in which the Globus Project has thrived, while seeding commercial grids based on open standards.

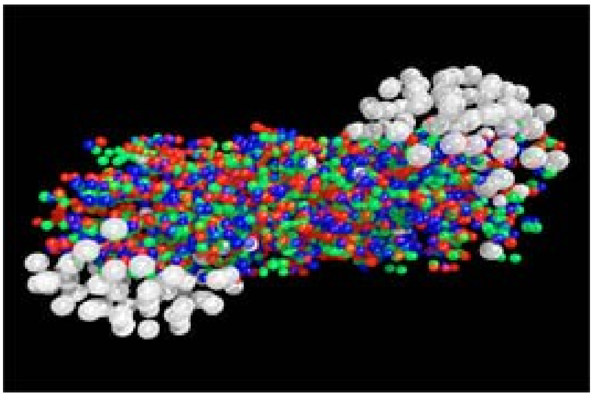
2004 saw the formation of Univa Corporation, a company devoted to providing commercial support for Globus software, and 2005 the creation of the Globus Consortium by a group of companies with an interest in supporting Globus Toolkit enhancements for enterprise use.

3.1.4. Examples of the Globus Alliance's Impact

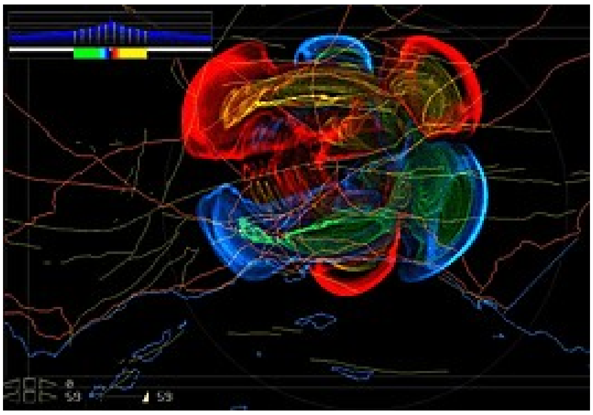
The Globus Alliance and the Globus Toolkit have enabled many exciting new scientific and business applications. The images here showcase just a few of the advances that have been helped by Globus technology.


© 2005 The University of Chicago / Argonne National Laboratory

Computational scientists at Brown University are using the Globus Toolkit and MPICH-G2 to simulate the flow of blood through human arteries. This image, prepared at Argonne National Laboratory, shows velocity (red arrows) and pressure (surface color) within a branched, three-dimensional arterial structure. The simulation was conducted using Nektar (software developed at Brown University) and was the first high-performance simulation to run in a distributed fashion using systems at multiple TeraGrid sites.

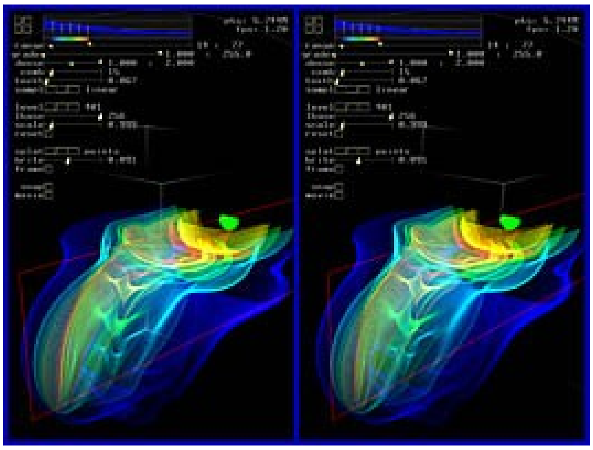


Globus Toolkit driven grid computing is central to management of large datasets generated by colliders such as those at CERN. This simulation shows two colliding lead ions just after impact, with quarks in red, blue, and green and hadrons in white.

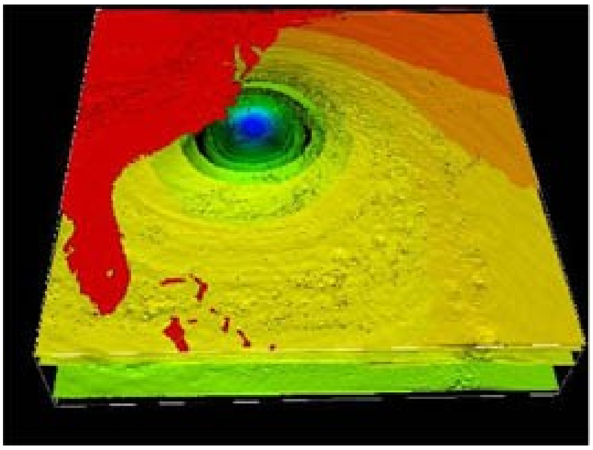


The Southern California Earthquake Center uses Globus software to visualize earthquake simulation data. Scientists simulate earthquakes by calculating the effect of shock waves as they propagate through various layers of a geological model. SCEC simulations cover a very large space with very high resolution and can generate up to 40TB of data per simulation run. This image shows earth movement from San Joaquin Valley CA, to Mexico, across the Los Angeles basin, moments after a simulated rupture. Blue and red lobes depict motion in opposite directions caused by shock waves along the fault.

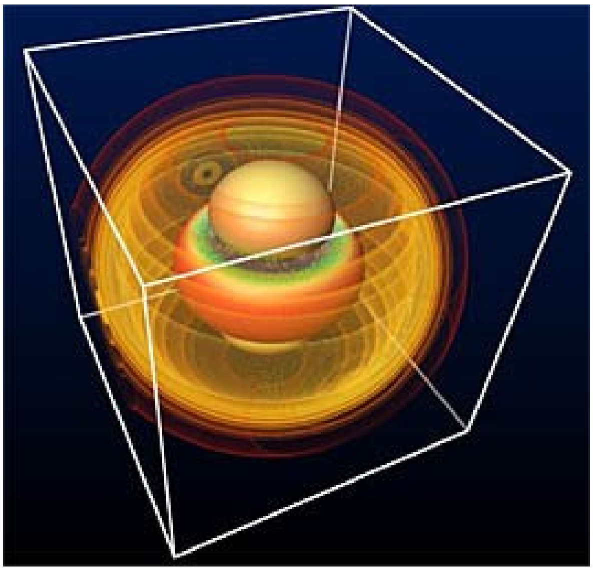
Scientists in the National Fusion Collaboratory are learning to use the Access Grid and Globus web services to participate remotely in pulsed plasma fusion experiments. The remote interface provides sensor readings, data analysis, audio, and video available in the control room and allows the team to discuss what is happening. The Access Grid is integrated with grid services and applications using the Globus Toolkit's security and communication libraries.



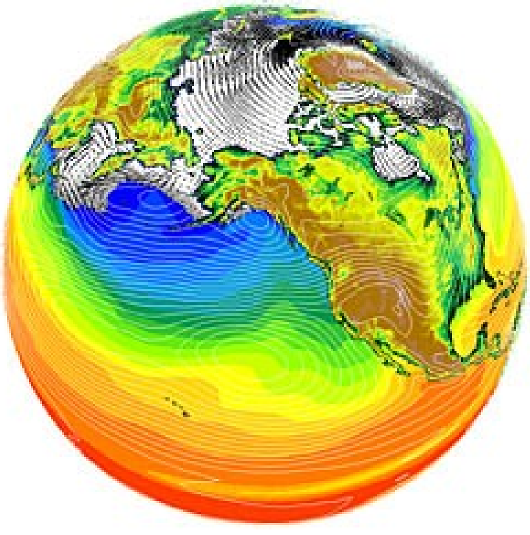
The Southern California Earthquake Center uses Globus software to visualize earthquake simulation data. Scientists simulate earthquakes by calculating the effect of shock waves as they propagate through various layers of a geological model. This image shows the velocity of motion from a simulated earthquake through a uniform earth. The image is provided in stereo so that it can be viewed in 3D.



The Globus Toolkit supports scientific data visualization on the TeraGrid. This image is part of a sequence that reveals the progression of Hurricane Isabel in September 2003. A desktop application uses the Globus Toolkit to launch parallel visualization tasks on multiple TeraGrid graphics nodes. The application allows users to connect to and interact with these tasks. Data provided by the National Center for Atmospheric Research.



Physicists used the Globus Toolkit and MPICH-G2 to harness the power of multiple supercomputers to simulate the gravitational effects of black hole collisions. The team, which included researchers from Argonne National Laboratory, the University of Chicago, Northern Illinois University, and the Max Planck Institute for Gravitational Physics in Germany, was awarded a prestigious Gordon Bell prize for its work. Image courtesy of Max Planck Institute for Gravitational Physics.



Scientists in the Earth System Grid (ESG) are producing, archiving, and providing access to climate data that advances our understanding of global climate change. This image displays data from ESG and shows sea ice extent (white/gray), sea ice motion, sea surface temperatures (colors), and atmospheric sea level pressure (contours). ESG uses Globus software for security, data movement, and system monitoring. Image provided by UCAR.

3.2. Instrument Middleware Project

Instrument Middleware [IM] is working on several projects to make scientific instruments and sensors more accessible and to integrate them in to data and compute grids. The Common Instrument Middleware Architecture (CIMA) project, supported by the National Science Foundation Middleware Initiative, is aimed at "grid enabling" instruments as real-time data sources to improve accessibility of instruments and to facilitate their integration into the grid. The main motivations of the Common Instrument Middleware Architecture (CIMA) project are to provide methodology for interacting with instruments and sensors in real-time from grid applications. Moreover, some abstraction of sensor and instrument functionality is needed to make grid applications that use them more robust and flexible. Additional, Virtual organizations that share instruments and sensors need location, authentication, and authorization mechanisms. In reality, Data collection needs to be interactive. Results of data analysis may be used to decide whether more data needs to be collected or what other data needs to be collected, and these cannot be achieved if instruments are kept offline. The figure below shows a simple 2-D analysis of instrument taxonomy.

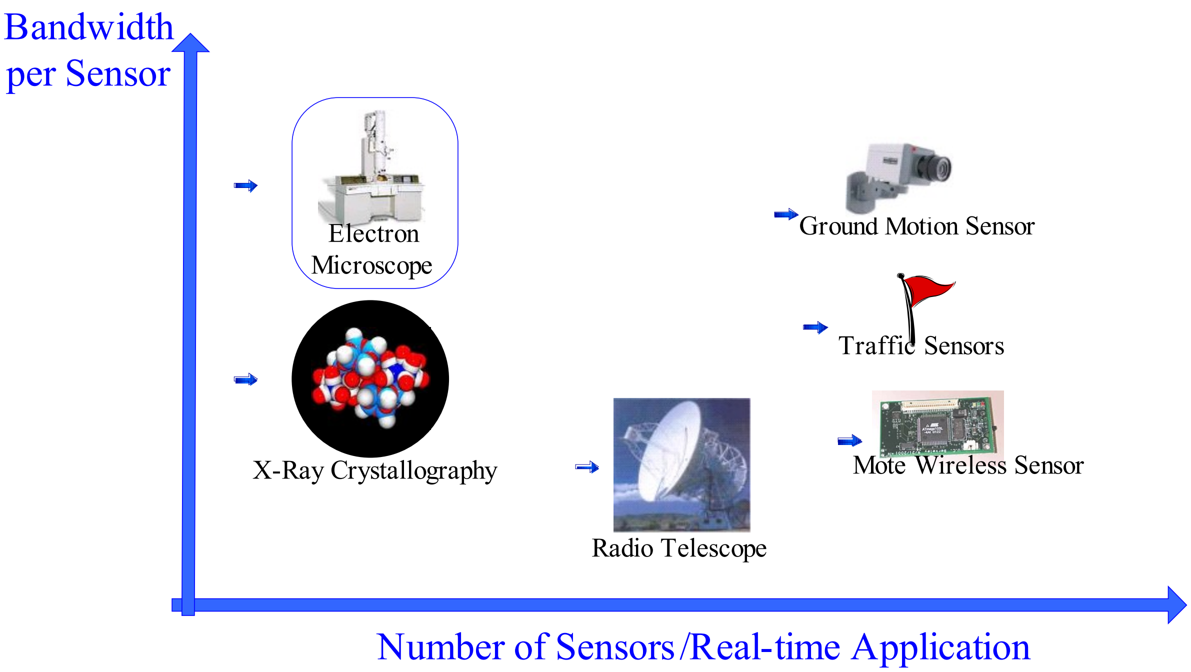


Figure 3.2.1: A simple 2-D analysis of instrument taxonomy.

General notions of the Instrument Middleware project are in order to improve instrument utilization, better integration with compute and storage elements, address control aspects of instruments, reduce overhead for users, reach a broader set of users, and standard interface methodology across a broad range of instrument types. Integration of instruments/sensors into the rest of the processing pipeline implies (as figure 3.2.2 shows):

- A uniform mechanism for locating and interacting with individual instruments or groups of sensors
- A scheme for maintaining investments in acquisition and analysis software as sensor design and engineering evolves

- A straightforward mechanism for fusing data from several types of sensors, i.e. leveraging sensor hardware investments by different groups in cooperative research

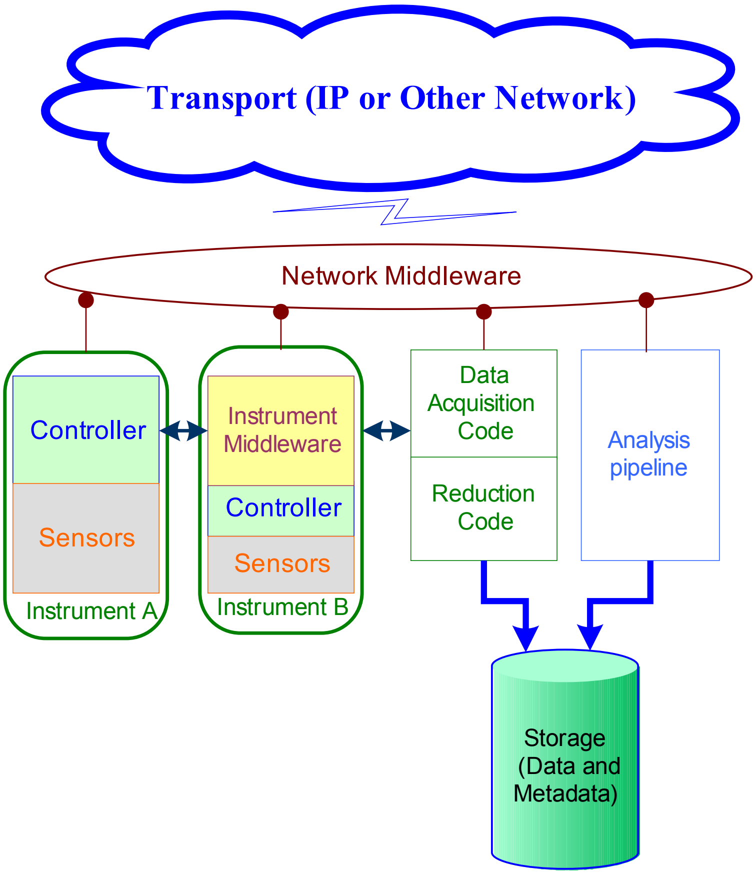


Figure 3.2.2: Integration of instruments/sensors into the processing pipeline.

The Instrument Middleware Project encompasses several projects aimed at improving access to scientific instruments and sensors.

CIMA. CIMA middleware is based on platform independent standards such as web services. Emphasis is placed on supporting a variety of instrument and controller types. CIMA promote interdisciplinary collaboration by facilitating compatibility between applications and instruments.

X-Ray Crystallography. The IU Molecular Structures Center is a major collaborator in CIMA. The goal is to remotely manage data collection from a high-intensity beamline located at a facility like the Argonne Advanced Photon Source

MMSF Automated Observatory. CIMA Applications in robotic telescopes and astronomy.

3.2.1. Common Instrument Middleware Architecture

The daily work of researchers in many scientific and engineering fields is dominated by the need to handle large-scale data. While ten years ago number-crunching resources were the limiting factor, now the ability to represent, locate, organize, securely access, and move data is often the bottleneck in the overall scientific process. This is exacerbated by new opportunities for sharing and collaborating which the Web and its associated tools provide. Scientists increasingly want to have the same ready access to distributed resources and each other that current desktop tools provide in business computing. The situation is further complicated by the need for many groups in global collaborations to analyze and add value to raw data collected from many sites with similar functions, e.g. synchrotron X-ray sources, electron microscopes, telescopes, HEP experiments, etc. Furthermore, the increasing speed and sophistication of wide-area networks and storage systems has outpaced the ability of software to effectively utilize distributed data from many sources. The problem: How to bring instruments, sensors and other real-time data sources into a grid computing and storage environment. The Common Instrument Middleware Architecture (CIMA) project, supported by the National Science Foundation Middleware Initiative, is aimed at "grid enabling" instruments as real-time data sources to improve accessibility and throughput in instrumentation investment and to facilitate their integration into the grid computing environments through a Service Oriented Architecture. CIMA middleware is based on current grid implementation standards and accessible through platform independent standards such as the Open Grid Services Architecture (OGSA) and the Common Component Architecture (CCA). Emphasis will be placed on supporting a variety of instrument and controller types including creating a small implementation that can be used with tiny wireless controllers such as the Berkeley Mote sensor package, as well as embedded PC-104 and VME-based controller systems. The CIMA project is addressing basic issues in sensor interaction design to simplify remote access, tele-presence and data management for instrument facilities. CIMA puts instruments and sensors "online" in a standards based way using web services. Web services interfaces can be wrapped in WS-RF to provide these as grid services. Also, to do these while collaborating with scientists in academia and industry in a broad range of disciplines who either develop instruments or whose work depends on the details of using them. CIMA-enabled instruments fit naturally into the scientific workflow and portals provide a useful "GUI" for real-time instrument-driven experiments based on CIMA. CIMA is being used to bring users and instruments together through a federation of Indiana, Midwest and international X-ray diffraction labs.

Rationale. Scientific instruments and sensors are crucial to scientific advancement. They provide the raw observations used to develop, verify, and falsify theories. Data from instruments typically have an extensive lifecycle, which includes corrections and calibration, annotation, and then storage in a database or file system. Researchers are working on every transitional data phase listed above, except for the earliest stage where the instrument provides its data as output, e.g. while database systems have had large scale convergence on a few API standards, instruments vary widely in their architecture, construction, and external interfaces. The Instrument Middleware Project proposes a single virtualization layer to hide this complexity, and present a relatively simple web service interface to the rest of the data pipeline[IM].

Architecture. The figure below illustrates the components of a typical instrument (blue background) and the added Common Instrument Middleware Architecture (yellow background). The instrument consists of a sensor or sensors (bottom) connected to a controller package, typically a CPU, memory and analog to digital or digital to analog converters. In addition to the hardware there is programming to implement basic functions such as setting parameters or taking readings (the "native call interface"). The controller package communicates with the world through an interface using a protocol. To this CIMA (in yellow) adds uniform ways of determining the characteristics of the instrument and interacting with it through W3C web services. The Proteus communications library allows the instrument to select the most appropriate protocol to talk to the data acquisition application [IM].

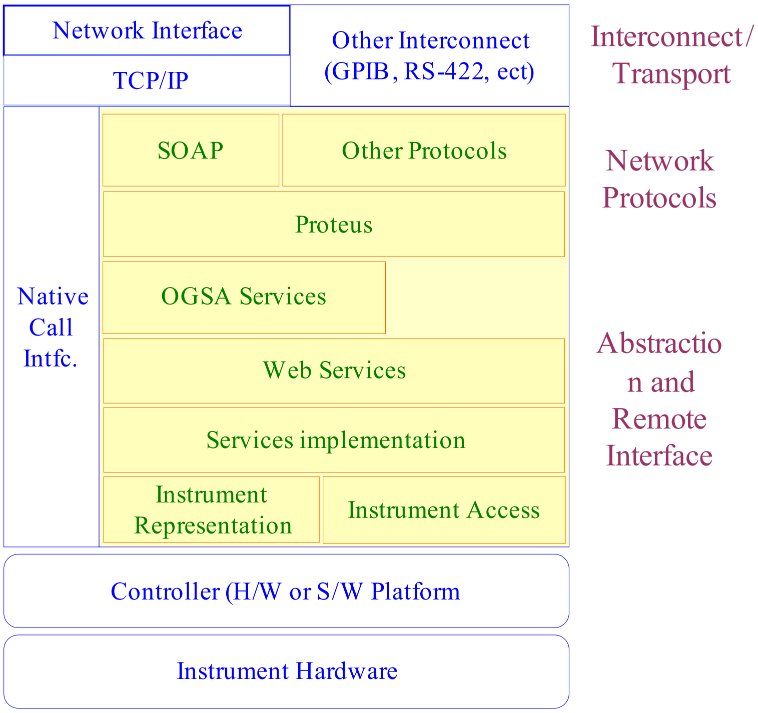


Figure 3.2.3: CIMA architecture.

Common Instrument Middleware Architecture Elements. Service implementation works for accessing the instrument's functionality and metrics using industry standard service implementation methodology (web services). There are two components: one is Plug-in modules are used to interface to hardware and the other one is Channel service that provides a network interface via web services to WS-RF grid service. Functions to register with a directory service (e.g. UDDI), authenticate users, provide access control to instrument controls and data, and co-schedule the instrument into a grid computing and storage context. Schemata for representing instrument functionality in WSDL and OWL-DL with instances (RDF) built into the instrument; Model for representing instrument function, metrics, and calibration. A small, high performance web services stacks (Java and C++) including Proteus support for multiprotocol, multimodal transport.

Applications. CIMA implementation two types of targets:

- Large scientific instruments — very large systems, few elements (e.g. Synchrotron beamline, APS/ALS)
- Embedded sensors and controllers — very small systems, many elements (e.g. PC104 industrial controller board, MICA Mote wireless sensor/controller board).

Currently, CIMA are applying to a range of instrument types, including crystallography using high brilliance X-ray sources, network performance monitors, and ultra-small wireless sensors such as the MICA MOTE. The CIMA abstraction layer reduces the dependence of acquisition and analysis software on the details of the hardware design, and facilitates the integration of instruments into the grid. Instrument users can reuse existing software when instruments are modified or upgraded. By offering CIMA interfaces to their products, instrument manufacturers can take advantage of a broader range of control application and data sinks (data fusion and analysis software). CIMA also facilitates interaction between research groups working at different places in the value chain of scientific inquiry.

3.2.2. X-Ray Crystallography

Several applications were chosen to evaluate CIMA design and implementation. One of the more fully developed applications is in X-ray crystallography [Mc05a].

In 1950's, precession cameras were used to take undistorted pictures of the "reciprocal lattice". Each point in the reciprocal lattice corresponds to a plane within the crystal. Images of the reciprocal lattice obtained from Precession photographs were used to determine the symmetry and cell dimensions for the crystal. Weissenburg cameras were used to measure intensities of each of the reciprocal lattice points. Typical exposure of several hours to several days that a few dozen to perhaps over 100 films per crystal. A Distorted view of the reciprocal lattice is obtained in each Weissenburg image, and each of the spots seen corresponds to a particular plane within the crystal. Intensities were estimated visually using an intensity scale obtained by taking different exposure times for a given reflection.

In 1970's, Single-crystal Diffractometers greatly increased the speed and accuracy of data collection. The crystal and a detector were positioned so that the intensity of diffraction from each plane in the crystal could be measured automatically. The single crystal diffractometer increased the accuracy of measurements by probably an order magnitude, and allowed much larger (and more complex) structures to be studied. Although it was computer controlled, it was limited by the serial nature (i.e. it could only measure one data point at a time). Typical crystals could require anywhere from a few days to months to be completely characterized

In 1990's, Bruker-AXS SMART 6000 CCD system located in the Indiana University Molecular Structure Center at Indiana University. The CCD detector consists of a fluorescent screen and a fiber optic bundle to increase the area being surveyed. In this view of a Bruker SMART6000 CCD the 4K x 4K CCD chip is located behind the circular beryllium window. The operator's console will allow the researcher to orient the crystal and view the resulting CCD images.

CIMA applications in crystallography labs and at synchrotron light sources. CIMA is used for remote monitoring of X-ray crystallogrpahy instruments in labs and at synchrotron light sources. The software architecture is shown in the figure above. There are four main components:

1. The Instrument Representative that implements the CIMA interface between the diffractometer CCD detector and several sensors related to it,
2. MyManager, a data manager that provides location-independent storage of CCD frames and sensor data from the detector,
3. A web service, the Data Manager Web Service (DM-WS) through which applications can access data acquired from the data manager, and
4. A portal and set of application-specific portlets that use the DM-WS to find and present data.

CIMA can be used to create a federation of labs with complementary capabilities sharing instruments and expertise. In the figure below three labs share a common portal and computing infrastructure for analysis.

In the current lab federation the Indiana University Data Capacitor is used as a high speed high capacity network data cache to receive detector CCD frames from all labs in the federation. The cached frames are then processed in to the MyManager data management system shown in the figure above, and metadata entries are created to help users find their data on multiple storage facilities.

The figure below shows schematically how labs are typically provided with a CIMA proxy service that sends data from the lab's diffractometer and other sensors in the lab. This streaming data consisting of CCD frames and lab environmental conditions is received by a data manager hosted by IU (or possibly by the lab itself) and is stored first in a short term sample cache and later archived on long term storage media. A portal and crystallography-specific portlets for viewing and managing data is used by crystallographers and sample providers to track the acquisition of data and to review the quality of the data during collection [Mcm06].

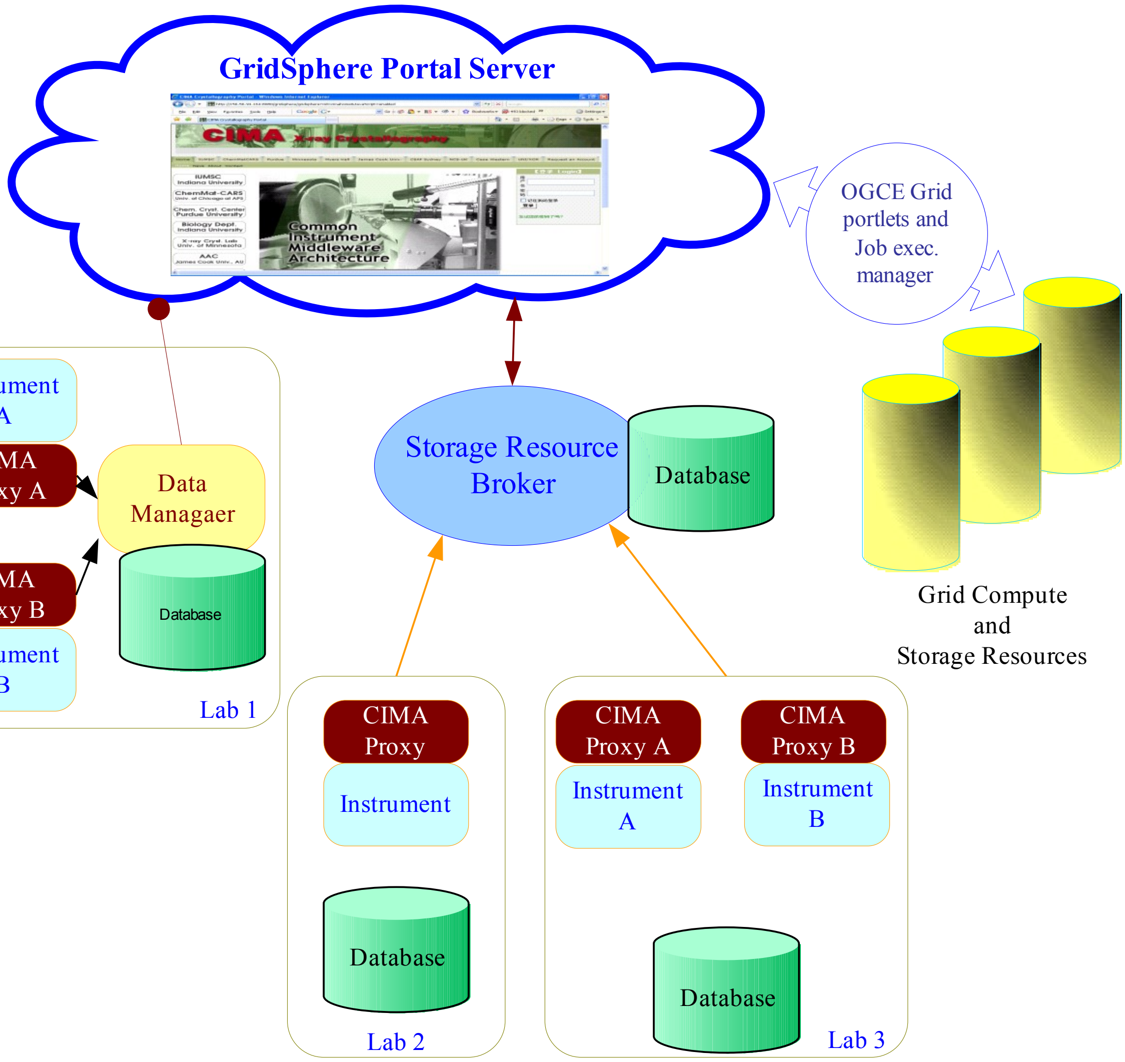

GridSphere Portal Server
CIMA X-ray Crystallography
IUMSC Indiana University
ChemMatCARS Univ. of Chicago at APS
Chem. Cryst. Center Purdue University
Biology Dept. Indiana University
X-ray Cryst. Lab Univ. of Minnesota
AAC James Cook Univ., AU
Common Instrument Middleware Architecture
OGCE Grid portlets and Job exec. manager
Grid Compute and Storage Resources
Instrument A
CIMA Proxy A
Data Managaer
CIMA Proxy B
Database
Instrument B
Lab 1
Storage Resource Broker
Database
CIMA Proxy
Instrument
Database
Lab 2
CIMA Proxy A
CIMA Proxy B
Instrument A
Instrument B
Database
Lab 3

Figure 3.2.4: Typical CIMA applications in X-ray crystallography.

The GridSphere portal shown in figure 3.2.5 illustrates instrument sharing and remote access to a group of CIMA-enabled diffraction labs.

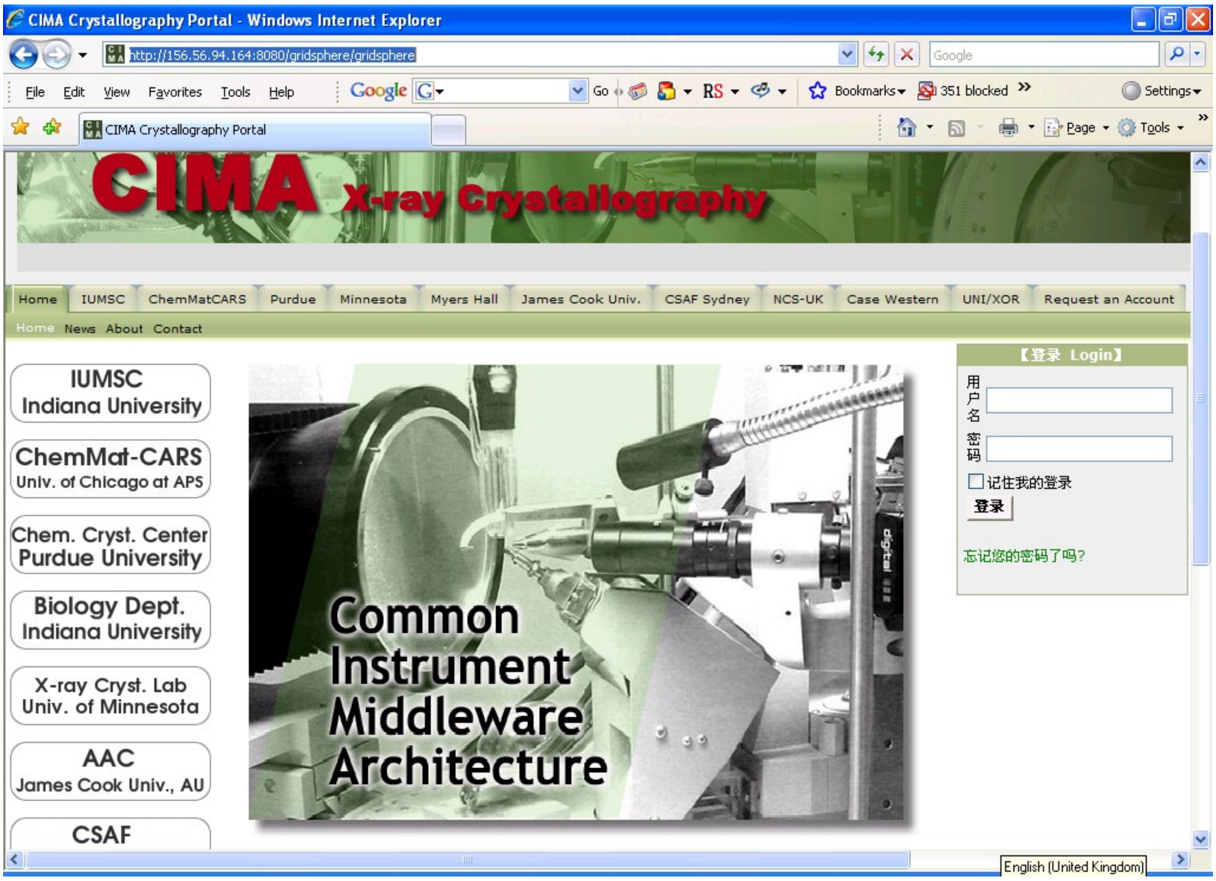

CIMA Crystallography Portal - Windows Internet Explorer
http://156.56.94.164:8080/gridsphere/gridsphere
File Edit View Favorites Tools Help
Bookmarks
351 blocked
Settings
CIMA Crystallography Portal
Page
Tools
CIMA X-ray Crystallography
Home
IUMSC
ChemMatCARS
Purdue
Minnesota
Myers Hall
James Cook Univ.
CSAF Sydney
NCS-UK
Case Western
UNI/XOR
Request an Account
Home News About Contact
IUMSC Indiana University
ChemMat-CARS Univ. of Chicago at APS
Chem. Cryst. Center Purdue University
Biology Dept. Indiana University
X-ray Cryst. Lab Univ. of Minnesota
AAC James Cook Univ., AU
CSAF
Common Instrument Middleware Architecture
【登录 Login】
用户名
密码
记住我的登录
登录
忘记您的密码了吗?
English (United Kingdom)

Figure 3.2.5: Remote access to a group of CIMA-enabled diffraction labs.

3.2.3. MMSF Automated Telescope

CIMA Applications in robotic telescopes and astronomy. The Morgan-Monroe Station (MMS) of the Geothe Link Observatories is situated in the Morgan-Monroe State Forest (MMSF) about 12 miles north of the Indiana University campus and is moderately protected from light pollution. The system has 33 distinct "sensors", 12 controllers. The MMS site houses a 16-inch automated telescope known as RoboScope and a new 50-inch automated telescope known as SpectraBot. These are active research facilities that operate most clear nights, and both are used for photometric studies. Each telescope has a separate dome. RoboScope is devoted to long-term monitoring of cataclysmic variable stars and related objects, typically obtaining one or two 4-minute exposures of about 100 objects every clear night. SpectraBot will be devoted to automated long-term monitoring (both CCD imaging and spectroscopy) of time-variable sources. The guide camera attached to it may also be used for other scientific observations. Roboscope has been in near-continuous usage since 1990 and Spectrabot is still being developed. The MMS is valuable for CIMA because it provides a large range of instruments, from weighing scales for a liquid N2 container to CCD cameras producing FITS formatted image fields [NM]. The system architecture is shown in figure 3.2.6.

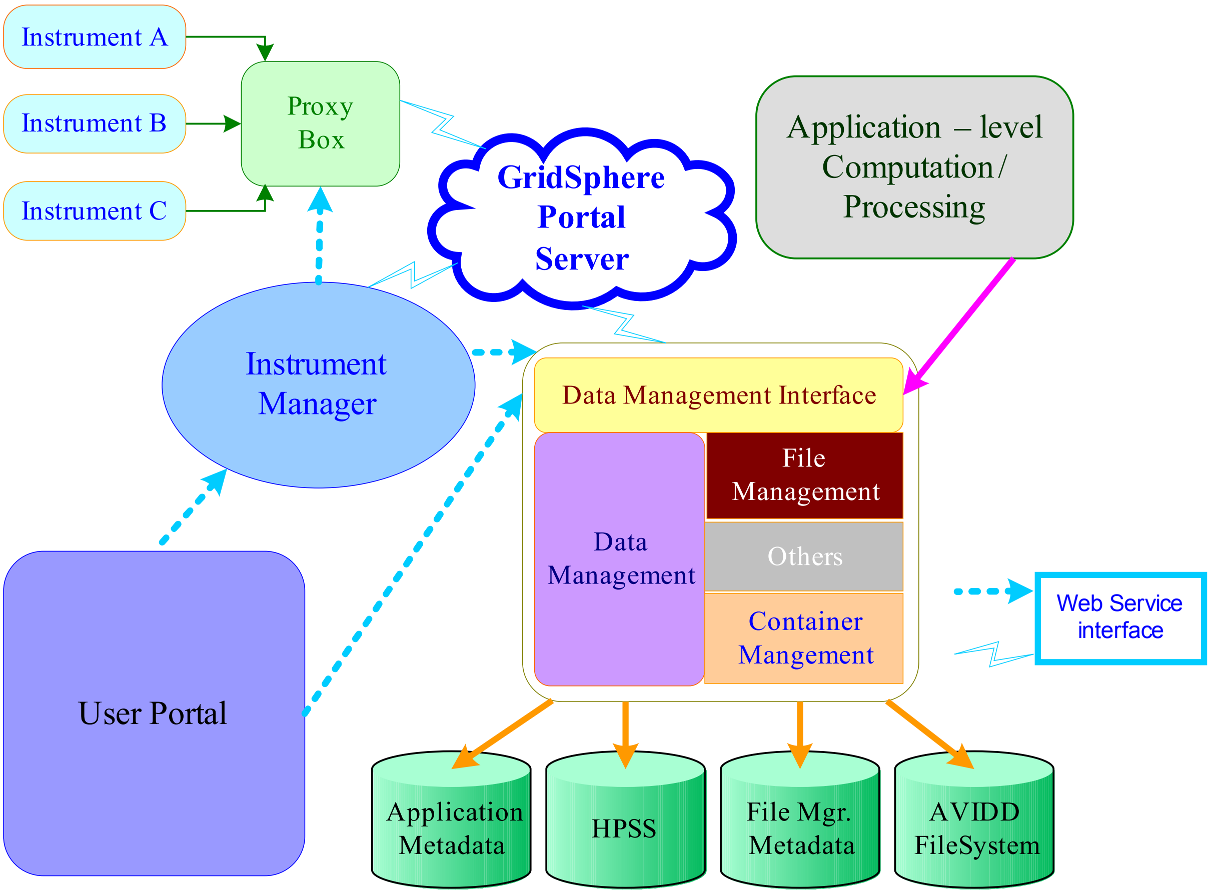


Figure 3.2.6: Data architecture for MMS and crystallography.

3.2.4. References

[Bos03] D. Bosio, W. H. Bell, D. Cameron, G. McCance, A. P. Millar et al, EU DataGrid Data Management Services. UK e-Science All Hands Conference, September 2003.

[Cha06] Ruay-Shiung Chang; Jih-Sheng Chang; Adaptable Replica Consistency Service for Data Grids. Third International Conference on Information Technology: New Generations, 2006. pp646–651, April 2006

[Cig06] M. Ciglan, M. Babik, L. Hluchy,: Services for replica consistency handling in data grids. Submitted to Physics of Particles and Nuclei Letters, published by the Joint Institute for Nuclear Research, Dubna, ISSN 1814-5957, 2006

[Dul01] D. Dullmann, W. Hoschek, J. Jean-Martinez, A. Samar, H. Stockinger, K. Stockinger: Models for Replica Synchronisation and Consistency in a Data Grid, 10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10'01), 2001.

[Fos04] I. Foster and C. Kesselman. The Grid2: Blueprint for a new computation infrastructure, Morgan Kaufmann, 2004

[FTS] http://www.gridpp.ac.uk/wiki/GLite_File_Transfer_Service

[IM] The Instrument Middleware Project, http://www.instrument-middleware.org

[Mc05a] D. F. McMullen, K. Huffman, "Connecting Users to Instruments and Sensors: Portals as Multi-user GUIs for Instrument and Sensor Facilities", In Proceedings of GCE 2005: Workshop on Grid Computing Portals held in conjunction with SC05, Seattle, WA, Nov. 18, 2005. Submitted to the Journal of Concurrency and Computation: Practice and Experience.

[Mc05b] D.F. McMullen, Chiu, K., "CIMA: Scientific Instruments as First Class Members of the Grid", Molecular and Materials Structure Network Workshop, March 28, 2005, Marysville, Australia.

[Mcm06] D.F. McMullen, "CIMA and Semantic Interoperability for Networked Instruments and Sensors." Open Collaboration: Networking Semantic Interoperability Across Distributed Organizations and Their Ontologies, Workshop at the National Science Foundation, Washington, DC, August 15, 2006. PPT

[NM] NSF Middleware Initiative, http://www.nsfmiddleware.org

[RTF] http://www.globus.org/grid_software/data/rft.php

[Ste06] G.A. Stewart, G. McCance, Grid Data Management: Reliable File Transfer Services' Performance, CHEP06, Mumbai, India, February 2006

[Sun04] Yuzhong Sun and Zhiwei Xu, Grid Replication Coherence Protocol, 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA, pp.232-239 April 2004.

[Vaz02] S. Vazhkudai, J.M. Schopf and I.Foster, Predicting the performance of wide area data transfers Proceedings of International Parallel and Distributed Processing Symposium, IPDPS' 02,pp34–43, 2002

[Wan06] Hsiangkai Wang; Pangfeng Liu; Jan-Jan Wu; A QoS-Aware Heuristic Algorithm for Replica Placement. 7th IEEE/ACM International Conference on Grid Computing, pp96–103, 2006

3.3. Common Instrument Middleware Architecture

The Common Instrument Middleware Architecture (CIMA) is the product of the Instrument Middleware Project for integrating instrumentation with the grid and making instruments/sensors a first-class citizen within the grid ecosystem. It is a framework for making instruments and sensors network accessible in uniform way, and for interacting remotely with instruments and the data they produce. Essentially, CIMA is a set of standards and software components, which are researched and developed towards the aforementioned goal. It is based on the emerging Open Grid Services Architecture (OGSA) and makes extensive use of newer technologies and standards such as WSRF, OWL-DL, and RDQL. CIMA is analogous to the GridCC effort of the European Union, although it follows a slightly different approach by customizing instrument interfaces with respect to the specific instruments, while GridCC has a uniform instrument interface.

3.3.1. Requirements and Design Goals

Grid computing has brought a degree of coherence to the development effort in large-scale computing projects, and has enhanced the ability to focus distributed computing and storage resources on a single large-scale application. Instruments, however, have been either treated on an ad hoc basis or are proxied by the files they generate.CIMA aims to bring instruments within the grid as first class participants via a middleware architecture that interoperates both syntactically and semantically with grid standards, yet still satisfies the performance, compatibility, and reliability demands of the underlying hardware it mediates.

CIMA addresses the following requirements [McMullen05]:

Functional Transparency. The grid interfaces must completely and accurately represent each function of the instrument. Grid applications must be able to develop a complete operational model of the instrument from minimal knowledge.

Resource-oriented stateful services. Instrument control and acquisition details are typically represented procedurally in the instrument's API, or in the user interface of control application. Representational State Transfer (REST is a hybrid style derived from several network-based architectural styles and combined with additional constraints that define a uniform connector interface) provides an alternative to a procedural API that is more interoperable, extensible, and scalable. The REST approach defines a small number of operations such as the HTTP actions PUT and GET on a large number of resources (e.g., URLs in the analogy to HTTP). OGSI has adopted some REST tenets in the form of Service Data Elements (SDEs). CIMA uses SDEs or resource property constructs to expose an instrument's characteristics as metadata so that an application can discover not only the existence and network address of an instrument, but also the channels provided to the application, their meaning, metrics, and use.

Protocol independence. Though SOAP is used as the common protocol, it is not appropriate for every situation. Performance is modest and may be limiting, so middleware must be able to use other protocols. Also, it may be the case that the middleware is connected to the instrument via a wire protocol of some kind other than IP, in which case, a standard way of incorporating vendor-specific protocols into the acquisition software is needed, without making the acquisition software vendor-specific.

Efficient communication. For situations requiring high communication throughput, the variety of message-exchange patterns provided by message-oriented middleware may be more appropriate than a remote procedure call approach. For the most demanding applications even messages may incur too much overhead since each message is typically treated as a datagram and separately routed in the middleware layer. In situations where each message is small, but the rate is high a more efficient approach is to use datachannels modeled after the virtual circuit concept. Data is streamed through the channel, with much of the per datum overhead shifted to the relatively infrequent channel set-up and tear-down steps.

CIMA wishes to be usable for all kinds of equipment, from large-scale scientific apparatus, down to small standalone sensors. Significant to the design of CIMA is the difference between the two similar, but distinct scenarios: Remote access and distributed operation. Remote access allows a scientist working off-site to access the instrument. Distributed operation, on the other hand, is a more profound development in scientific instruments. In distributed operation, the functions of the conventional instrument site itself are distributed within a virtual organization. By applying the appropriate grid technologies, each user can interact with the instrument as if she were the sole user, and explore innovations that in a conventional, centralized setting would have an unacceptable impacton other users.

This requirement for distributed operation sets the following design goals [Devadithya05]:

Boot-strappable. A central design requirement is that CIMA applications must be able to develop an operational model of the instrument from a minimum of external knowledge, which requires that each function of the instrument is completely and accurately described. This requirement will encourage the kind of loose-coupling that promotes inter- operability, thus reducing the burden ofmanaging and administering a large variety of instruments.

Interoperable. In scientific collaborations one research group would need access to grid-enabled instruments maintained by another group. In order to achieve this, the specification of a sensor should be complete enough so that third party applications can access it without additional information. Also, minor changes to sensor functionality should not require deep code changes. Making the functionality independent of the data structures as much as possible would increase interoperability.

Efficient data transfer. Some instruments and sensors, especially when aggregated, may generate data at high rates. If the data rate is higher than the rate at which the system can transfer them, then there could be data loss or system crashes. Even though buffering could be used to handle mismatches between the data rates for a short period, it will not be possible to operate indefinitely.

Lightweight. Sensors may need to be deployed at locations subject to electrical and processing power constraints. For example, a seismic sensor located underwater in deep sea will have all these constraints in addition to bandwidth limitations. While it is unlikely that a computer will be associated with each of such sensors, a computer should be present as closely as possible. The computer located in such a remote area may have limited processing power and memory. Therefore, CIMA implementations should require a minimum of computing, storage, and network resources. Although the usual limiting resource is power, network bandwidth constraints or intermittent connectivity may create secondary requirements for additional short term or persistent storage at the sensor. This creates tradeoffs in memory allocation between data buffers and program address space.

Support for intermediaries: Intermediaries are important for signal processing or buffering functions between a sensor and the consumer (figure 3.3.1). This off-loads the work of serving multiple consumers from the sensor to an intermediary. The intermediary can have one input stream from the sensor and multiple output streams for the consumers. Intermediary also can act as security gateways for the sensor node by allowing only particular intermediaries to connect to it. Also, filters can be implemented at the intermediate nodes.

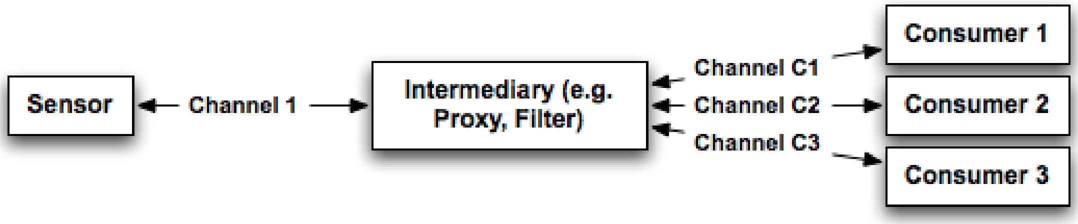


Figure 3.3.1: Intermediary between sensor and consumers

3.3.2. Approach

In order to cover for the requirements and design goals elaborated upon in the previous paragraph, CIMA has followed a modular, hierarchical approach. The following subsections provide more details on the exact approach [Devadithya05].

Layered specification. To provide reusability and interoperability of instrument interfaces, CIMA strives for layered specifications. For example, a lower-layer specification corresponding to a pressure sensor should be reusable with that corresponding to a temperature sensor, with minimal modifications. Then an application written for one sensor would have a fair degree of functionality (i.e., require minimal code changes) even with another sensor. The ultimate goal of this approach is to promote the reuse of code components between applications.

Plug-ins. While the sending and receiving of data is consistent among different types of instruments, different instruments and sensors may require specific processing to read; and the construction and interpretation of data messages would be specific to each of them. Plug-ins are used to perform these specific functions at the data source as well as at the data consumer. An intermediary that needs to perform some processing of the message, such as calculate the average of data values and send only the result would also need to have a plug-in to perform such tasks.

Loose coupling. Loose-coupling encourages interoperability by minimizing the dependencies between the system components, such as sensor, data consumer, and intermediaries. This is achieved by implementing a document-oriented message-passing model. Each message, whether data or control, would be an XML document containing the data along with some meta-data required to interpret them.CIMA calls this message a "parcel".

Ontology. One shortcoming of instruments and sensors is that the applications that use them (e.g., data acquisition codes) must have a complete operational model of the instruments and sensors they work with built-in as lines of code. This makes maintaining investments in these codes difficult and expensive when the underlying instrument hardware is improved. A primary design goal for this project is to externalize the instrument description so that applications can build an operational model "on the fly". This approach makes it possible to preserve investments in data acquisition codes as instrument hardware evolves, and to allow the same code to be used with several similar types of instruments or sensors. This is particularly important in situations where the instrument or sensors and the related acquisition and analysis codes are in their early stages of development and undergoing rapid change.

Hierarchical. Instruments are hierarchical in nature. Client applications are simplified if they can access one stream of data as opposed to multiple streams. This could be achieved if the top level instrument can aggregate the data from lower level instruments. The parent instrument should be able to provide information about its children, such as their interfaces and data rates, to client applications. In CIMA, instruments may be arranged hierarchically where a parent instrument is considered to be composed of multiple child instruments in a nested manner, with no limit to the depth of nesting or to the actual location of the child components.

Push and pull models. Data messages may be "pulled" on demand at the cost of a request-response cycle, or they can be directly "pushed" when scheduled (or as available) from the sensor to the receiver using one-way messages. Both models are useful, depending on the requirements, with the pull model usually being more convenient, and the push model more efficient. Since the push model does not require a request-response, multiple messages can be batched into a single call. CIMA supports both models. In the push model, the consumer has to maintain an endpoint to which the sensor can stream-out data. A pull-model is more suitable if the consumer is only interested in receiving the current reading from the sensor.

3.3.3. Architecture

Instrument model. CIMA's instrument model is shown in figure 3.3.2. An instrument consists of one or more sensors. Each sensor may serve zero or more consumers. A consumer can receive data from one or more sensors. The communication between a sensor and a consumer forms a virtual link, which is called a channel. An instrument is allowed to aggregate data from several sensors and send them via one channel. This is shown in the communication between Instrument A and Consumer A in figure 3.32. This approach is largely similar to the GridCC one. In the latter case, the sensors are handled by the Instrument Element's (IE) Instrument Managers (IMs), in a similar hierarchical fashion. However, in GridCC, the control plane is distinguished from the data plane, and they use separate communication channels. As it is shown in figure 3.3.3, a single (architectural) channel is enclosing all traffic, while in GridCC the two channels are architecturally distinct.

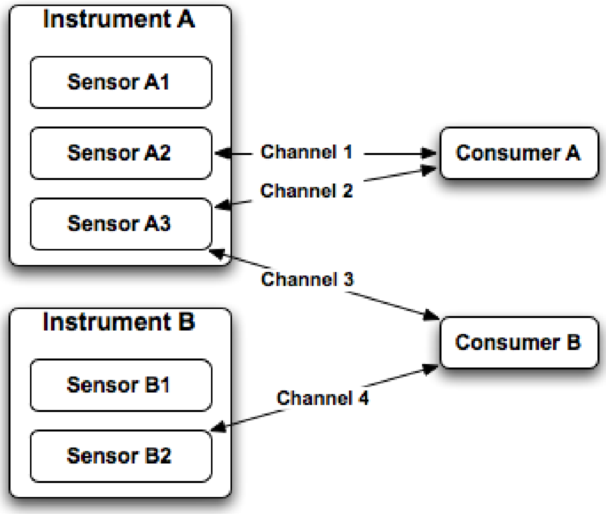


Figure 3.3.2: Instrument model.

Channels and plug-in modules. The application consists of channels and plug-in modules. A channel provides a generic framework for the communication while the plug-ins implement application-specific functionalities. As shown in figure 3.3.3, the channel handles the communication between the sensor and the data consumer. Specific plug-in modules, both at the sensor-end and the consumer-end, implement the sensor-specific behavior. The Channel has two modes of operation, namely the Source (sensor-end) and the Sink (consumer-end). Each mode runs a grid service instance for receiving messages (control and data) from the other. The Channel Source's grid service mainly handles control information from the data consumer, such as registering with the sensor to receive sensor data and un-registering to stop receiving data. It also responds to one-time requests for sensor data. The grid service at the Channel Sink receives streaming data and status messages such as "sensor data not available" from the sensor. A data consumer can choose to receive a single data value (request-response or pull model) or continuously receive data values (streaming or push model). In the pull model, the consumer sends a request for sensor data and the sensor responds with the current value of its data. However, in push model, first the consumer registers with the sensor to receive data, indicating the data rate required and the port number on which it wants the data to be sent to. The sensor then starts a thread, which will continuously poll its sensor data at the requested rate and send them to the requested port at the consumer. The thread will continue to run until it receives an un-register request from the consumer or after a given number of attempts to send data fails.

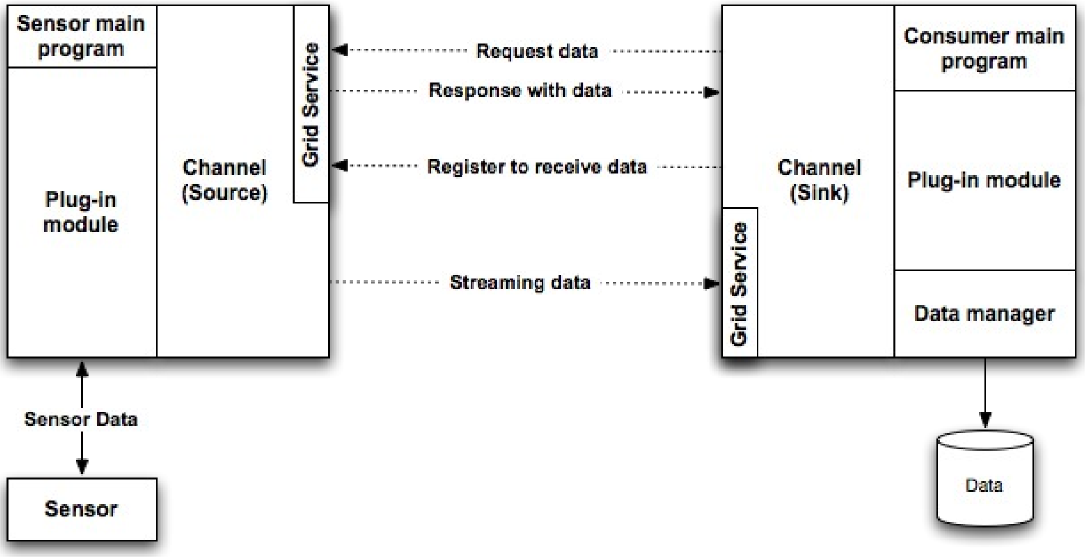


Figure 3.3.3: Communication via a channel.

Communication protocols. SOAP is currently used, since it is the most widely accepted standard for web services. The current implementation of the channel uses gSOAP to handle the serialization/deserialization of SOAP messages and the communication. While by default, gSOAP (the main toolkit used by CIMA) uses HTTP as the transport layer, prototype systems that use Antelope [Antelope] and Binary XML for Scientific Applications (BXSA) have been developed as HTTP alternatives.

Parcel. Different sensors generate data in different formats. To address this issue, CIMA came up with the "parcel" concept. A parcel may contain the following elements:

- **Type** is a URN that uniquely identifies the type of the parcel. Application-level parcel handlers will recognize the type of the parcel, and unwrap it. Register message and temperature data would be examples for this field.
- **ID** of the parcel is given as a URI.
- **Location** indicates where the actual data is located. If the data is contained within the parcel, it is indicated as inline.
- **Encoding** of the data helps recipients to interpret them. Binary is one encoding.

- **Body** contains the actual parcel data (if the location is inline).

All the fields except those for location and body are optional. Parcels encapsulate a description which makes consumers capable of performing the data acquisition.

Sensor ontology. A key objective of the CIMA approach is to make the instrument or sensor self-describing and to push the production of meta-data about what the instrument is producing as far toward the instrument as possible. The former objective, self-description, assists components downstream in the data acquisition and reduction process to understand and manage the instrument or sensor effectively. The latter objective, annotating the data coming from an instrument, provides information needed for proper handling of the data. The development of these components is based on a CIMA ontology for instruments and sensors, which is based on the OWLDescription Logic formalism.

3.3.4. References

[McMullen05] D. McMullen, T. Devadithya, K. Chiu; "Integrating Instruments and Sensors into the Grid with CIMA Web Services"; Proceedings of the 3rd APAC Conference on Advanced Computing, Grid Applications and e-Research (APAC05)

[Devadithya05]T. Devadithya, K. Chiu, K. Huffman, D. McMullen; "The Common Instrument Middleware Architecture: Overview of Goals and Implementation"; Proceedings of the 1st International Conference on e-Science and Grid Computing (e-Science'05)

[Antelope] http://www.brtt.com/

3.4. The Storage Resource Manager, Broker and the Internet Backplane Protocol

3.4.1. The SDSC Storage Resource Broker

The Storage Resource Broker (SRB), developed at the San Diego Supercomputer Center (SDSC), supports shared data collections that can be distributed across multiple organizations and heterogeneous storage systems. The SRB can be used as a Data Grid Management System (DGMS) that provides a hierarchical logical namespace to manage the organization of data (usually files).

The SRB software infrastructure can be used to enable Distributed Logical File Systems, Distributed Digital Libraries, Distributed Persistent Archives, and Virtual Object Ring Buffers. The most common usage of SRB is as a Distributed Logical File System (a synergy of database system concepts and file systems concepts) that provides a powerful solution to manage multi-organizational file system namespaces.

SRB presents the user with a single file hierarchy for data distributed across multiple storage systems. It has features to support the management, collaboration, controlled sharing, publication, replication, transfer, and preservation of distributed data. The SRB system is middleware in the sense that it is built on top of other major software packages (file systems, archives, real-time data sources, relational database management systems, etc). The SRB has callable library functions that can be utilized by higher level software. However, it is more complete than many middleware software systems as it implements a comprehensive distributed data management environment, including end-user client applications ranging from Web browsers to Java class libraries to Perl and Python load libraries [1].

3.4.2. Data Grids

Data grids support massive terabyte scale data collections, with possibly millions of files, that are distributed across multiple institutions. Several data management systems are based upon a generic data management infrastructure, the Storage Resource Broker.

The management of data has traditionally been supported by software systems that assume explicit control over local storage systems (file systems) or that assume local control over information records (databases). The SRB manages distributed data, enabling the creation of data grids that focus on the sharing of data, digital libraries that focus on the publication of data, and persistent archives that focus on the preservation of data. Data grid technology provides the fundamental management mechanisms for distributed data. This includes support for managing data on remote storage systems, a uniform name space for referencing the data, a catalog for managing information about the data, and mechanisms for interfacing to the preferred access method. Digital libraries can be implemented on top of data grids through the addition of mechanisms to support collection creation, browsing and discovery. The underlying operations include schema extension, bulk metadata load, import and export of metadata encapsulated in XML, and management of collection hierarchies. Persistent archives can be implemented on top of data grids by addition of integrity metadata needed to assert the invariance of the deposited material. The mechanisms provided by data grids to manage access to heterogeneous data resources can also be used to manage migration from old systems to new systems, and hence manage technology evolution. The Storage Resource Broker is being used as the underlying infrastructure for both digital libraries and persistent archives, and is a proof in practice that common infrastructure can be used for data management.

The data grid community defines "data" to be the strings of bits that comprise a digital entity. A digital entity might represent, for example, a data file, an object in an object ring buffer, a record in a database, a URL, or a binary large object in a database. Data are stored in storage repositories (file systems, archives, databases, etc.). Meaning is assigned to a digital entity by associating a semantic label. Information consists of the set of semantic labels that are assigned to strings of bits. The semantic labels can be used to assert a name for a digital entity, assert a property of a digital entity, and assert relationships that are true about a digital entity. Information is stored in information repositories (relational databases, XML databases, flat files, etc.). The combination of a semantic label and associated data is treated as metadata. Metadata are organized through specification of a schema and stored as attributes in a relational database. The digital entities that are registered into the database comprise a collection. The metadata in the collection in turn provides the context for interpreting the significance of the registered digital entities.

Grids manage distributed execution of processes. The SRB data grid manages simulation results, observational data, and derived data products. Grids and data grids are complementary technologies that together enable the creation and management of data. Digital libraries organize information in collections. Persistent archives preserve the information content of collections. Persistent archives manage the evolution of all components of the hardware and software infrastructure, including the encoding syntax standards for data models [2].

3.4.3. The Internet Backplane Protocol

The Internet Backplane Protocol (IBP) is middleware for managing and using remote storage. It was invented to support Logistical Networking in large scale, distributed systems and applications. We define logistical networking as the global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources. This contrasts with more traditional networking, which does not explicitly model storage or computation resources in the network. We call this approach "logistical" because of the analogy it bears with the systems of warehouses, depots and distribution channels commonly used in the logistics of military and industrial activity. IBP provides a mechanism for using distributed storage for logistical purposes.

It got its name because it was designed to enable applications to treat the Internet as if it were a processor backplane. Whereas on a typical backplane, the user has access to memory and peripherals and can perform direct communication between them with DMA, IBP gives the user access to remote storage and standard Internet resources (e.g. content servers implemented with standard sockets) and can perform direct communication between them with the IBP API.

By providing a uniform, application-independent interface to storage in the network, IBP makes it possible for applications of all kinds to use logistical networking to exploit data locality and more effectively manage buffer resources. We believe it represents the kind of middleware needed to overcome the current balkanization of state management capabilities on the Internet, so that any application that needs to manage distributed state can benefit from the kind of standardization, interoperability, and scalability that have made the Internet into such a powerful communication tool [3].

3.4.4. Storage Resource Managers

A collaboration involving the EU DataGrid, Jefferson Lab, Fermi Lab and the Lawrence Berkeley National Laboratory is developing middleware components called Storage Resource Managers (SRMs). Storage Resource Managers (SRMs) are middleware software modules whose purpose is to manage in a dynamic fashion what resides on the storage resource at any one time. SRMs do not perform file movement operations, but rather interact with operating systems, mass storage systems (MSSs) to perform file archiving and file staging, and invoke middleware components (such as GridFTP) to perform file transfer operations. There are several types of SRMs: Disk Resource Managers (DRMs), Tape Resource Managers (TRMs), and Hierarchical Resource Managers (HRMs). We explain each next. Unlike a storage system that allocates space to users in a static fashion (i.e. an administrator's interference is necessary to change the allocation), SRMs are designed to allocate and reuse space dynamically. This is essential for the dynamic nature of shared resources on a grid.

Disk Resource Managers (DRM) manages dynamically a single shared disk cache. This disk cache can be a single disk, a collection of disks, or a RAID system. The disk cache is available to the client through the operating system that provides a file system view of the disk cache, with the usual capability to create and delete directories/files, and to open, read, write, and close files. However, space is not pre-allocated to clients. Rather, the amount of space allocated to each client is managed dynamically by the DRM. The function of a DRM is to manage the disk cache using some client resource management policy that can be set by the administrator of the disk cache. The policy may restrict the number of simultaneous requests by each client, or may give preferential access to clients based on their assigned priority. In addition, a DRM may perform operations to get files from other SRMs on the grid. This capability will become clear later when we describe how DRMs are used in a data grid. Using a DRM by multiple clients can provide an added advantage of file sharing among the clients and repeated use of files. This is especially useful for scientific communities that are likely to have an overlapping file access patterns. One can use cache management policies that minimize repeated file transfers to the disk cache for remote grid sites. The cache management policies can be based on use history or anticipated requests.

Tape Resource Managers (TRM) provide a middleware layer that interfaces to systems that manage robotic tapes. The tapes are accessible to a client through fairly sophisticated Mass Storage Systems (MSSs) such as HPSS, Unitree, Enstore, etc. Such systems usually have a disk cache that is used to stage files temporarily before transferring them to clients. MSSs typically provide a client with a file system view and a directory structure, but do not allow dynamic open, read, write, and close of files. Instead they provide some way to transfer files to the client's space, using transfer protocols such as FTP, and various variants of FTP (e.g. Parallel FTP, called PFTP, in HPSS). The TRM's function is to accept requests for file transfers from clients, queue such requests in case the MSS is busy or temporarily down, and apply a policy on the use of the MSS resources. As in the case of a DRM, the policy may restrict the number of simultaneous transfer requests by each client, or may give preferential access to clients based on their assigned priority.

Hierarchical Storage Managers (HRM) are a TRM that have a staging disk cache for its use. Thus, it can be viewed as a combination of a DRM and a TRM. It can use the disk cache for pre-staging files for clients, and for sharing files between clients. This functionality can be very useful in a data grid, since a request from a client may be for many files. Even if the client can only process one file at a time, the HRM can use its cache to pre-stage the next files. Furthermore, the transfer of large files on a shared wide area network may be sufficiently slow, that while a file is being transferred, another can be staged from tape. Because robotic tape systems are mechanical in nature, they have a latency of mounting a tape and seeking to the location of a file. Pre-staging can help mask this latency. Similar to the file sharing on a DRM, the staging disk in an HRM can be used for file sharing. The goal is to minimize staging files from the robotic tape system.

The concept of an SRM can be generalized to the management of multiple storage resources at a site. In such cases, the site SRM may use "site-file-names" (directory path + file names) which do not reflect the physical location and file names. This gives the site the flexibility to move files around from one storage device to another without the site-file-names changing. When a client accesses a file using a site-file-name, it may be given in response the physical location and file name. The client can then use the physical file name to execute a file transfer.

In general, it is best if SRMs are shared by a community of users that are likely to access the same files. They can be designed to monitor file access history and maximize sharing of files by keeping the most popular files in the disk cache longer [4].

3.4.5. References

[1] The San Diego Supercomputer Center Storage Resource Broker (SRB), http://www.sdsc.edu/srb

[2] Digital Libraries and Data Intensive Computing, R. Moore, China Digital Library Conference , Beijing, China, September 2004

[3] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In NetStore '99: Network Storage Symposium. Internet2, October 1999.

[4] Storage Resource Management Working Group, http://sdm.lbl.gov/srm-wg/

3.5. GridFTP Middleware

The GridFTP protocol provides for the secure, robust, fast and efficient transfer of (especially bulk) data. The Globus Toolkit provides the most commonly used implementation of that protocol. GridFTP provides support for parallel data transfer using multiple TCP streams between the source and the destination. GridFTP also provides support to transfer data that is striped across multiple hosts by using one or more TCP streams between m hosts on the sending side and n hosts on the receiving side. GridFTP allows an authenticated third-party to initiate, monitor and control a data transfer between storage servers. Checkpointing is used to provide fault tolerance. A failed transfer is restarted from the last checkpoint. It extends the partial transfer mechanism defined in the standard FTP to support transfers of arbitrary subsets of a file. GridFTP also allows manual or automatic control ofTCP buffer size.

As mentioned previously, GridFTP uses TCP as the underlying transport mechanism. Therefore, TCP implementation and configurations will have a fundamental impact on communication performance [5].

3.5.1. References

[5] GridFTP, http://www.globus.org/toolkit/data/gridftp/

3.6. Gridge Toolkit

Gridge Toolkit is an open source software initiative developed by PSNC aimed to help users to deploy ready-to-use grid middleware services and create productive grid infrastructures. All Gridge Toolkit software components have been integrated together and form a consistent distributed system following the same interface specification rules, license, quality assurance and testing.

Gridge Toolkit components have been successfully tested with different versions of Globus Toolkit TM as well as other core grid middleware solutions. The Gridge Toolkit software is available for free with a full commercial support. Additionally to the following services PSNC offers for its partners and users:

- technical support, consulting, training and development for Gridge Toolkit and Globus Toolkit,
- assistance in design, deployment and configuration of grid middleware software,
- on-site installation and integration of Gridge Toolkit and Globus Toolkit key components,
- workshop and hands-on training on "grid enabled" technologies.

3.6.1. Tools and services

Gridge Toolkit consists of the following tools and services:

- GridSphere Portal Framework (developed within Grid-Lab)
- Grid Service Provider (GSP) (developed within Progress)
- Grid Resource Management System (developed within GridLab)
- Grid Authorization Service (developed within GridLab)
- Grid Mobile Services (developed within GridLab)
- Grid Data Management System (developed within Progress)
- Migrating Desktop (developed mainly within CrossGrid and continued in other projects).
- Grid Monitoring System (Mercure) (developed within GridLab)
- System level checkpointing library

All the pieces are integrated with each other and follow the same interface specification rules, license, quality assurance and testing, distribution, etc. The most important tools and services are described in the next subsections.

3.6.2. Grid Service Provider

The user access philosophy in Gridge is drawn around the concept of enabling multiple independent user access applications, such as web portals, standalone applications and mobile user interfaces for the utilization by the users of the grid infrastructure. In this philosophy each user may execute his grid work using any of the available user interfaces and may be doing any part of his work using any of these interfaces. Such an approach required a special attention concerning the quick delivery of important grid data such as application descriptions or job configurations and statuses, and concerning the construction of a single point of entry to many independent and sometimes also heterogeneous grid environments. This motivation led to the design of the Grid Service Provider (GSP) module. GSP is a set of high level grid services whose primary task is to support various types of grid user interfaces in quick and seamless access to the data and information flowing in from the lower level services. Thanks to the introduction of GSP these data and information can be easily shared between heterogeneous user interfaces without having direct access to the lower level services. For example, it is not necessary to contact a grid execution engine to check the configuration of running grid jobs or to check their status: all this information is stored in the GSP database and can be quickly read and delivered to a user. GSP contains two high level services: the Job Submission Service and the Application Management Service. The Job Submission Service delivers functions for computing job building, submitting them to the grid for execution and viewing the results. It allows to create jobs, configure their tasks and set the requirements for grid environment resources. The Job Submission Service features the grid resource broker plug-in mechanism, which allows it to cooperate with multiple independent grid infrastructures, thus allowing the users to submit exactly the same job to two different grid environments. The task of a grid resource broker plug-in is to communicate with the grid execution engine in the communication protocol used by that engine and to translate the job structure into the language used by that engine. For example, a plug-in for the Gridge Resource Management System is familiar with GRMS access interface and with the XRSL language used by that service. The Application Management Service manages the Gridge application repository that can also be shared between various independent user interfaces. An application descriptor contains a reference to the application's executable and a set of its available, required or optional arguments, required environment variables as well as input and output files. One executable may be referenced by many applications, and different application configurations are viewed as independent applications.

Both GSP services support workflows: The Job Submission Service allows to configure workflow jobs and send them for the execution in the grid, and the Application Management Service allows to create descriptors for workflow applications and use them as the base for the creation of new job configurations.

3.6.3. Grid Portal

In Gridge, the portal access to grid services is organized with the use of the GridSphere portal framework. Created by the Portals Work Package in the GridLab Project, GridSphere leverages the most relevant standards, best-practices and technologies to offer a framework for developing grid portals. One of the most exciting standards to gain adoption by the general community is the Portlet Java Specification Request (JSR 168). The Portlet JSR defines an application programming interface (API) and model for packaging and presenting Web content as portlets. Portlets are Java classes that have a clearly defined interface and life cycle. Portlets are hosted by a portlet container and can be presented in a Web page in any manner supported by the portlet container. The Portlet JSR makes it possible to distribute and share Web applications more easily, creating a means for collaborating on Web portal development on a much larger-scale. The GridSphere Project has also developed a generic framework for developing grid portal applications called Grid Portlets. Grid Portlets offers developers a collection of "portlet services" for performing tasks on the grid. These portlet services can be used to grid-enable any portlet web application. Grid Portlets provides a collection of simple, easy-to-use, well integrated portlets that showcase the functionality offered in Grid Portlets, including portlets for retrieving credentials, monitoring resources, submitting jobs and managing remote files. GridSphere can also be used as an environment to run specialized portlets cooperating with the services of the Grid Service Provider and of the Data Management System (DMS). These portlets, developed with the use of specially designed Portlet Framework that allows to run the portlets also in a standalone mode, utilize the high level functionality provided by GSP and DMS to organize more user-friendly access to grid services. The available portlets that allow to access the functionality of the GSP and DMS services include the "Applications" portlet, which allows to manage the Gridge application repository, "My computing Jobs" portlet, which allows to create and submit grid jobs on top of any application available in the repository and "My data" portlet, which allows to manage data files stored by the Data Management System. In addition to these core portlets, several specialized application portlets have been developed as examples of user friendly portal interfaces to grid applications. The main motivation behind the introduction of the Portlet Framework was to create a solution that supports developers of specialized user interfaces to different grid applications. Thanks to the architecture and technology used by the framework developers of specialized application portlets gain opportunity to create new portlets and enable new grid applications through easy-to-use job configuration wizards on the portal within several days. The Portlet Framework together with the GridSphere portal container and the Grid Service Provider provide a flexible set of tools to construct web-based grid access environment. In simple scenarios, involving usage of a limited number of simple applications, a GridSphere installation with the Grid Portlets fulfils the requirements. When the user access involves more sophisticated scenarios with multiple different applications utilized by multiple different user groups, the Grid Service Provider module accessed via the portlets created with the use of the Portlet Framework acts as a high level support for the administrators, developers and users.

3.6.4. GRMS

The Gridge Resource Management System (GRMS) is an open source meta-scheduling system, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. GRMS, based on dynamic resource selection, mapping and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic grid environment and resource management challenges, e.g. load balancing among clusters, remote job control or file staging support. Therefore, the main goal of GRMS is to manage the whole process of remote job submission to various batch queuing systems, clusters or resources. Finally, GRMS can be considered as a robust system which provides abstraction of the complex grid infrastructure as well as a toolbox which helps to form and adapts to distributing computing environments.

GRMS has been designed as an independent set of components for resource management processes. It can take an advantage of various low-level core grid services, such as e.g. GRAM, GridFTP and Gridge Monitoring System, as well as various grid middleware services, e.g. Gridge Authorization Service, Gridge Data Management Service and more. All these services working together provide a consistent, adaptive and robust grid middleware layer which fits dynamically to many different distributing computing infrastructures. The GRMS implementation requires Globus software to be installed on grid resources, and uses Globus Core Services deployed on resources: GRAM, GridFTP, MDS (optional). GRMS supports Grid Security Infrastructure by providing the GSI enabled web service interface for all clients, e.g. portals or applications, and thus can be integrated with any other middleware grid environment. One of the main assumptions for GRMS is to perform remote jobs control and management in the way that satisfies Users (Job Owners) and their applications requirements as well as constraints and policies imposed by other stakeholders, i.e. resource owners and grid or virtual organization administrators. All users requirements are expressed within XML-based resource specification documents and sent to the GRMS as SOAP requests over GSI transport layer connections. Simultaneously, Resource Administrators (Resource Owners) have full control over resources on which all jobs and operations will be performed by appropriate GRMS setup and installation. Note, that the GRMS together with Core Services reduces operational and integration costs for administrators by enabling grid deployment across previously incompatible cluster and resources. Technically speaking GRMS is a persistent service within a Tomcat/Axis container. It is written completely in Java so it can be deployed on various platforms. With the GAS, GRMS is able to manage both job grouping as well as jobs within collaborative environments according to predefined VO security rules and policies. With the Data

Management services from Gridge, GRMS can create and move logical files/catalogs and deal with data intensive experiments. Gridge Monitoring Service can be used by GRMS as an additional information system. Finally, Mobile service can be used to send notifications via SMS/emails about events related to users' jobs and as a gateway for GRMS mobile clients. GRMS is able to store all operations in a database. Based on this information a set of very useful statistics for both end users and administrators can be produced. All the data is also a source for further, more advanced analysis and reporting tools.

GRMS is composed of the following modules:

- **Job Receiver Module.** Provides GSI enabled web service interface for GRMS. In job submission this module is responsible for job description validation and putting proper job to a queue. For workflow jobs it creates graph representation of tasks and check its correctness.
- **Job Queue.** Stores jobs which are ready for execution. It is prepared for implementation of any queue management strategy and scheduling algorithms.
- **Broker Module** is the heart of GRMS. It steers the whole process of job submission: it gets jobs from queue, calls Resource Discovery Module to find appropriate resources, it evaluates resources to find "the best" one, it creates the environment for job execution by transferring input data, it calls the Job Manager Module to monitor status changes of a job; after the job is finished it takes care on transferring output data to location specified by a user. It also provides information about jobs and their history in the system.
- **Resource Discovery Module.** Finds resources that fulfil requirements described in Job Description. Resources are described in an XML document which contains parameters important from the job scheduling point of view. Resource Discovery Module can be configured to use many information sources. It can use e.g. Globus MDS, iGrid service, Mercury Monitoring Service, Adaptive Components Service, Testbed Information Service.
- **Job Manager Module.** Responsible for monitoring of status changes of job, and for job control: job cancelling, suspending and resuming.
- **Job Registry Module.** Responsible for storing all information about jobs, and make it available for other modules.

An XML based GRMS Job Description (GJD) language was specified to allow users defining the computing jobs and resource requests. For each task there is a section in a job description document describing resource requirements and user preferences used for dynamic resource discovery. Another section defines the application: executable, input and output files required, arguments, environment, etc. One of the most interesting features of GRMS is its ability to deal with jobs defined as a set of tasks with precedence relationships (workflows). With just one call a user can submit the whole computational experiment that consists of many independent application executions. Two ways of expressing the dependencies between tasks are possible. The first one is a direct way, based on the parent child relationship among tasks. In his case the execution of a child depends on the status change of its parents.

The second way of expressing dependencies is associated with a data flow between the tasks. Here a user can specify that an output of one task becomes the input for the other one. What is a beauty here is that a user does not have to specify the exact file locations. However, in such a case it is user's responsibility to define file dependencies correctly. GRMS will reject execution of the job where data dependencies contradict the parent-child relationship. As it was already mentioned the basic way of introducing dependencies between tasks is by defining the parent-child dependencies. A very interesting and novel feature of GRMS which distinguishes it from the other systems is that execution of a child task can be triggered by any status change of a parent task. So, not only a task termination can trigger following executions. This feature is very useful in many scenarios. For instance we can imagine that a user would like to execute some application as soon as the other one starts running — e.g. for client server communication. The other example could be the flow of computation that depends on the failure of execution of one of the tasks (failover mechanisms).

3.6.5. GAS

The Gridge Authorization Service (GAS) is an authorization system which can be the standard authorization decision point for all components of a grid system. Security policies for all system components can be stored in GAS. Using these policies GAS can return an authorization decision upon the client request. GAS has been designed in a way that makes it is easy to perform integration with external components and to manage security policies for complex systems. Full integration with the Globus Toolkit and many other grid services makes GAS an attractive solution for grid environments. As stated above an authorization service can be used for returning an authorization decision upon the user request. The request has to be described by three attributes: user, object and operation. The requester simply asks if the specific user can perform the operation on the specific object. Obviously, the query to an authorization service can be more complex and the answer given by such service can be complicated as well. By using the modular structure of GAS it is easy to write a completely new communication module. The GAS complex data structure can be used to model many abstract and real world objects and security policies for such objects. For example, GAS has been used for managing security policies for many Virtual Organizations, for services (like Gridge Resource Management Service, iGrid, Mobile Services and other) and for abstract objects like communicator conferences or HPC centres in Europe.

The main goal of GAS is to provide a functionality that would be able to fulfil most authorization requirements of grid computing environments. GAS is designed as a trusted single logical point for defining security policy for complex grid infrastructures. As flexibility is the key requirement, it is to be able to implement various security scenarios, based on push or pull models, simultaneously. Secondly, GAS is independent of specific technologies used at lower layers. It should be fully useable in environments based on grid toolkits as well as other toolkits. The high level of flexibility is achieved mainly through the modular design of GAS and efficient data model, with which one can define many scenarios and objects from the real world. It means that GAS can use many different ways for communication with external components and systems and many security data models and hold security policy on different types of storage systems. These features make GAS attractive for many applications and solutions (not only for those related with grids). GAS has to be the trusted component of each system in which it is used and brings about that the implementation of GAS was written in ANSI C. This choice makes GAS a very fast and stable component which consumes not much CPU power and little amount of memory.

The main problem of many authorization systems is their management. It is not easy to work with a complex system in a user-friendly way. Based on many experiences and the end user feedback the GAS administration portlet (web application) is provided, which makes management as easy as possible. Flexibility of this solution gives users a full possibility of presenting only these security policies which are important for them. The GAS management is possible in two other ways: by the GUI GTK client and by the command line client.

For Globus Toolkit users, GAS provides a set of plugins for Globus components, (for example: gatekeeper and job manager plug-in). These plug-ins communicate with GAS in a secure way and can ask GAS about an authorization decision.

3.6.6. Monitoring: Mercury

The Mercury Grid Monitoring System provides a general and extensible grid monitoring infrastructure. The Mercury Monitoring is designed to satisfy requirements of grid performance monitoring: it provides monitoring data represented as metrics via both pull and push access semantics and also supports steering by controls. It supports monitoring of grid entities such as resources and applications in a generic, extensible and scalable way. It is implemented in a modular way with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The aim of the Mercury Monitoring system is to support the advanced scenarios in a grid environment, such as application steering, self-tuning applications and performance analysis and prediction. To achieve this the general GGF GMA architecture is extended with actuators

and controls. Actuators are analogous to sensors in the GGF GMA but instead of gathering information, they implementcontrols and provide a way to influence the system.

The architecture of Mercury Monitor is based on the Grid Monitoring Architecture (GMA) proposed by Global Grid Forum (GGF), and implemented in a modular way with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The input of the monitoring system consists of measurements generated by sensors. Sensors are controlled by producers that can transfer measurements to consumers when requested. Sensors are controlled by producers that can transfer measurements to consumers when requested. Sensors are implemented as shared objects that are dynamically loaded into the producer at run-time depending on configuration and incoming requests for different measurements. In Mercury all measurable quantities are represented as metrics. Metrics are defined by a unique name such as host.cpu.user which identifies the metric definition, a list of formal parameters and a data type. By providing actual values for the formal parameters a metric instance can be created representing an entity to be monitored. A measurement corresponding to a metric instance is called metric value.

Metric values contain a time-stamp and the measured data according to the data type of the metric definition. Sensor modules implement the measurement of one or more metrics. Mercury Monitor supports both event-like (i.e. an external event is neededto produce a metric value) and continuous metrics (i.e. a measurement is possible whenever a consumer requests it such as, the CPU temperature in a host).

Continuous metrics can be made event-like by requesting automatic periodic measurements. In addition to the functionality proposed in the GMA document, Mercury also supports actuators. Actuators are analogous to sensors but instead of taking measurements of metrics they implement controls that represent interactions with either the monitored entities or the monitoring system itself.

3.6.7. Data Management System

Data storage, management and access in Gridge environment is supported by the Gridge Data Management Suite (DMS). This suite composed of several specialized components allows to build a distributed system of services capable of delivering mechanisms for seamless management of large amount of data. This distributed system is based on the pattern of autonomic agents using the accessible network infrastructure for mutual communication. From the external applications point of view DMS is a virtual file system keeping the data organized in a tree structure.

The usage of the DMS has been described in details in section 4.1 concerning VLab.

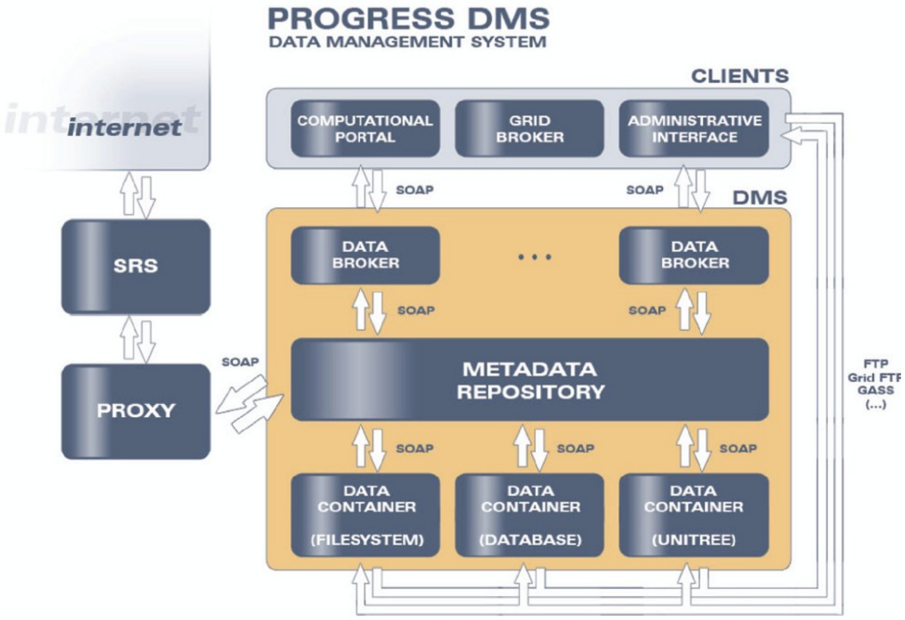


Figure 3.6.1: Gridge Data Management System architecture.

3.6.8. Mobile User Support

Mobile software development in Gridge (see figure below) is focused on providing a set of applications that would enable communication between Mobile devices, such as cell phones, Personal Digital Assistants (PDA) or laptops and grid services on the other side. This class of applications is represented by clients running on mobile devices, mobile gateways acting as a bridge between clients and grid services as well as additional specialized middleware services for mobile users. The main goal of the services is to make use of small and flexible mobile devices that are increasingly used for web access to various remote resources. The system provides grid access mechanisms for such devices. This requires adoption of the existing access technologies like portals for low bandwidth connectivity and low level end-user hardware. The mobile nature of such devices also requires flexible session management and data synchronization. The system enhances the scope of present grid environments to the emerging mobile domain. Utilizing new higher bandwidth mobile interconnects, very useful and previously impossible scenarios of distributed and collaborative computing can be realized. To achieve this and taking into consideration some still existing constraints of mobile devices, the Access for Mobile Users group is developing a set of applications in the client-server model with the J2ME CLDC/MIDP-Java client, and portlet server working with GridSphere. This set allows us to manage end user grid jobs (steer an application) or view messages and visualizations produced by grid applications on device such simple as standard mobile phone. The second group of developed services is tightly connected with end user notifications about various events in grids. Events like: the information about user application is started or finished, the visualization is ready for viewing or waiting for new data, can be sent to end users using various notifications way. It can be Email, SMS, MMS, or message of one of Internet Communicators like AIM, Yahoo, ICQ, Jabber etc. (including most popular in Poland Gadu-Gadu and Tlen). Mobile services gives also end users the possibility to start a conference concerning the aforementioned event between users of the given virtual organization (including conferences between clients of different communicators). The unique possibility of giving access to grid resources for users of relatively weak devices is one of features that distinguish Gridge mobile applications from other grid systems. Moreover, the used technology, Java 2 Micro Edition — Mobile Information Device Profile (J2MEMIDP) applications (midlets) on the client side allows to develop flexible, possibly off-line working programs that may be used on a wide range of devices supporting J2ME. Using the MIDP compliant device internal repository for storing data, gives the user possibility to use it later in offline state and prepare the data, to be sent in on-line state. The Mobile Command Center (MCC) that acts as a gateway between mobile client and grid services is developed in Java as a GridSphere portlet (see gridsphere.org) with separate "mobile" context. MCC automatically grabs the device profile (like device class, screen size, color depth, etc), this information is used during forwarding the request from mobile device to grid services (mainly GSI-enabled web services like Gridge MessageBox, Visualization Service for Mobiles or Gridge Resource Management System). Services that can be accessed from mobile device using MCC belong to two groups: the first group consists of grid services that were adopted to use with mobile devices, the second group are services developed for use only with mobile devices. The Visualization Service for Mobiles belong to second group and is used to view the application output in form of visualization prepared exactly according to the user's device capabilities. The advantage in this case is as follows: the large amount of data is not sent via weak GPRS connections to the device that cannot store it in the memory and cannot display it correctly. First group of services consists of Gridge Resource Management System and Notification and Messenger Service. The first service can be used in 'Collaborative scenario' — the user can steer the application (even not being an owner) from mobile device. He/she can get the jobs list, migrate, resume, suspend, cancel, edit, view history and submit new job on the basis of edited/modified description of already finished jobs. Using GRMS together with Notification service the user can register for user notifications related to the running jobs. In this way the user is notified about important events occurring in the grid (like jobs status changes, application output availability). These notifications can be sent as email, SMS or using an instant messenger (AIM, Yahoo, etc.) to the user. Using the messenger service it is possible also to make a conference between users of virtual organization defined in Gridge Authorization Service even if they use different communicators.

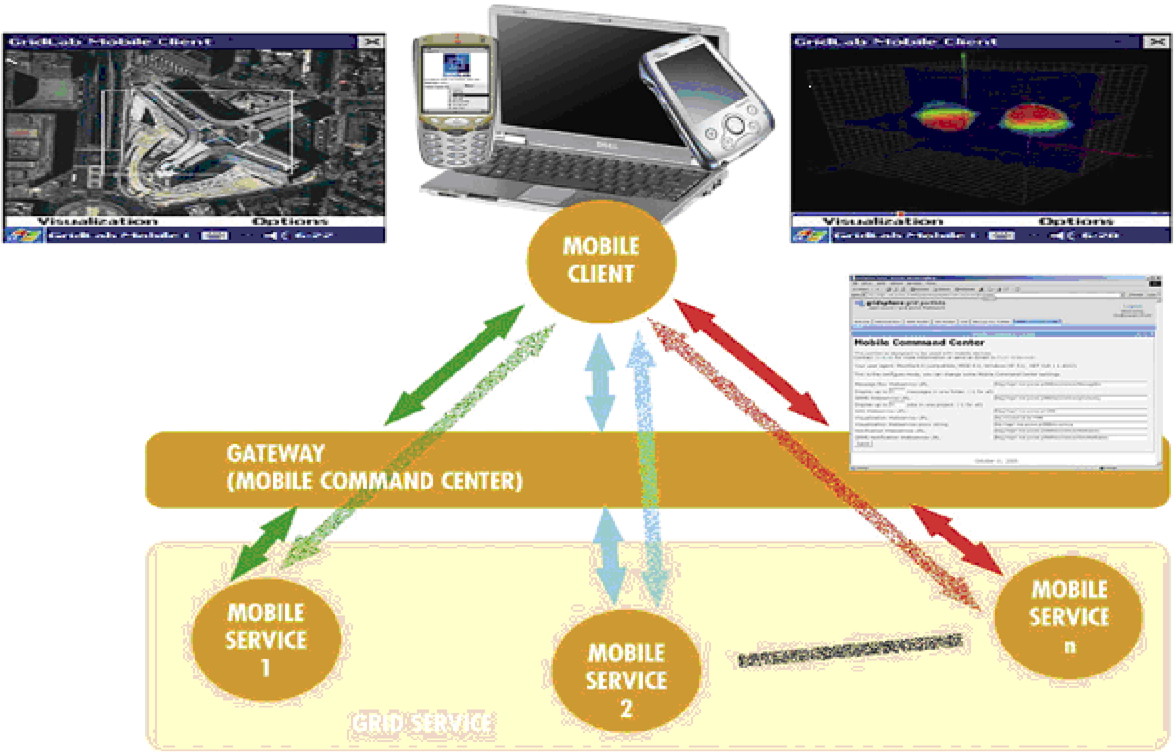


Figure 3.6.2: Mobile services architecture.

3.6.9. Accounting: VUS

Identification of users in any system is necessary for accounting and security reasons, e.g. in order to charge for used resources and tracing unfair behavior. On the grid level, the user is uniquely identified by the subject of his proxy certificate (the so called Distinguished Name, DN). The proxy may also contain some additional information related to the identification, like e. g. the name of a Virtual Organization on behalf which the user acts. On the other hand, on the operating system level, the user is identified by the user account, on which the processes performing user requests are run. Thus we face problem of mapping the global user identity (DN) to a local identity (account). The simplest solution is a 1-1 mapping, which means that the user must have a "personal" account on each node in the grid (this solution is implemented by the Globus gridmap file). This is not scalable: Bigger systems are hard to manage for obvious reasons. Another simple approach is the "n-1 mapping", where many users may be mapped to the same account. This is usually not sufficient, even if only users of the same organization are mapped to the same account. The mentioned accounting and security requirements are not fulfilled and moreover, the problem of unwanted interference of different users' jobs occurs.

The Virtual User System (VUS) addresses all these problems. VUS is an extension of Globus GRAM (gridmap callout) and allows running jobs without having a "personal" user account on a node. First, the user is authorized by querying set of authorization plugins. The example plugins are the gridmapfile (allows for backward compatibility with standard Globus mechanism) and GAS (allows for integration of VUS and GAS). The next step is the selection of a local account.

The "personal" accounts are replaced by "virtual" ones, which are mapped to users only for the time needed to fully process a job. The Account Manager assures that only one user is mapped to a particular account at any given time. The history of user-account mappings is stored in a database, so that accounting and tracking of useractivities are possible.

The local VUS database was designed to store both standard and non standard accounting data types. The standard accounting may be periodically gathered from the local accounting (operating system or local scheduling system level) and merged with global user identity by Accounting Module scripts. Then, the accounting may be published via web service interface.

3.6.10. Summary

In this section the grid toolkit called Gridge, developed by Poznan Supercomputing and Networking Center has been presented. Gridge can be used as a whole, including all the services and tools, or a user may choose just these services and tools that are important for specific scenario. Gridge provides a flexible, secure and robust grid infrastructure. It is currently being used in many grid infrastructures, including Clusterix (http://www.clusterix.pcz.pl), VLab (http://vlab.psnc.pl), InteliGrid (http://www.inteligrid.com), ACGT (http://acgt.ercim.org) and others.

3.6.11. References

[1] www.gridge.org

[2] http://gridlab.org

[3] GridSphere Project, http://www.gridsphere.org/

3.7. Visualization Middleware

With the advent of grid computing as a sensible tool for medium and large scale science over the last years the analysis of data produced by grid-enabled applications has become more and more crucial. One important aspect of this development is given by large-scale simulations in application domains like high energy physics, chemistry, or life sciences. This simulations typically produce output data, which requires suitable visualization tools for supporting the scientist in understanding the result of the simulation.

Therefore several approaches for grid-based visualization solutions have been developed. They can be differentiated based on what user community they address, as well as the underlying technology applied.

Solutions, which hide the inherent complexity of the grid from the scientific end user, provide good tools for visual analysis. A typical class of applications, which are fulfilling this requirement, are Problem Solving Environments (PSEs). Within these enviroments the scientist can design in-silico experiments interactively, which are subsequently executed on a computional grid. An example PSE is given by VLAM-G [AFSVLA], which enables the scientist to interactively construct experimental configurations. The system also allows remote control of the experiments conducted as well as collaboration mechanisms. Typical application domains are biosciences and material sciences.

Another important category of grid-based visualization solutions follows the remote visualization paradigm, which enables the user to observe a visualization which is executed on a remote grid resource. This approach enables the usage of powerful grid resources for rendering large-scale datsets while the user can observe the results from his local desktop computer. This approach is specifically sensible if the visualization can be executed on the same machine as the simulation or a machine nearby, thus reducing the amount of data which needs to be moved. An example integrated with OGSA-based webservices is presented in [BROAPP]. This approach applies the functionality of AccessGrid [CHIACC] for broadcasting the visualization, which is generated by the VISIT toolkit [EICSTE] to multiple remote observers. Another grid-based visualization solution, which is applying a portal-based user interface accessible with a web-browser, is presented in [JANDEP]. The visualization functionality provided by the backend server includes parallel volume rendering, while the results are stored as images, which can be investigated using the web portal. For the data transfer between the backend rendering system and the web portal Globus services [FOSGLO] are used. Within the Patras/ITBL system [SUZVIS] remote visualizations are delivered to a client computer over the grid for monitoring as well as steering a simulation executed on a supercomputer. It also offers the possibility of postprocessing visualization by connecting it to AVS/Express [UPSTHE]. The Visapult tool [BETGRI] visualizes the results of a grid-enabled general relativity simulation involving the collision of black holes and the emission of gravitational waves. It is using services provided by Cactus [ALLTHE] and Globus [FOSGLO] to access the grid. The rendering itself is done by a parallel direct volume renderer based on MPI. In [NORENA] an approach using local volume rendering while transmitting the volume over the grid is presented. The ordering of the volume for transmission is done in a progressive manner based on visibility, which is determined on the clients side. Furthermore the volume data gets wavelet compressed before transmitted over the grid. The Distributed Visualization (Dv) System [AESPRE] is a grid-based distributed visualization system mainly used for earthquake visualization. A central concept and the unit of work to be scheduled onto the grid is the active frame, a piece of data together with the program to be executed on it. The scheduling is done based on resource information delivered by the Remos [LOWDES] and Network Weather Service (NWS) [WOLTHE] systems. To improve the general usability of the system algorithms from AVS [UPSTHE] and the Visualization Toolkit (VTK) [SCHTHE] can be integrated. A distance tomography system operating on the grid and offering access to 3D patient data and visualization capabilities is described in [FOSDIS]. These visualization capabilities include geometry transmission for local rendering as well as video transmission of readily rendered images. In [CZAPRA] an approach for distributed visual exploration of large datasets using a data reduction pipeline is presented. The pipeline is distributed over the grid and data is transmitted during the investigation of the previous dataset. SGI graphics hardware is partially exploited to speed up the rendering process.

Regarding the integration of grid middleware into the visualization solution the approach presented in [NORENA] is rather loosely coupled with the grid middleware compared to the other grid-based approaches discussed above. In this solution the rendering is done at the client side with the volume data being progressively transmitted from the server where it was produced.

Considering the role of user interfaces for grid-based visualization solutions the situation has improved quite a bit over the last years in terms of usability. This is especially reflected by the usage of web portals or Java-based solutions as graphical user interfaces, which are both reducing the software requirements on the clients side to a minimum. This flexibility regarding the user interfaces can be provided rather easily since the grid middleware libraries to be integrated only provide basic services. A common user interface for grid applications is a web browser by using portal-based techniques such as in the approaches described in [JANDEP] and [SUZVIS]. An example for a Java-based grid user-interface is given by the Migrating Desktop (MD) [KUPMIG] which is supporting mobile usage due to its ability to be started from a web browser or Java Web Start.

3.7.1. General Considerations for Grid-based Visualization

A grid-based visualization solution commonly has to handle large scale input data. For such scenarios parallel rendering techniques as described in [CROPAR] are commonly well applicable. However within the heterogeneous grid environment the selection of appropriate visualization algorithms as well as the selection of resources is of significant importance. For supporting the design process of grid-based visualizations one can follow the reference model proposed by Haber and McNabb [HABVIS] as shown in figure 3.7.1.

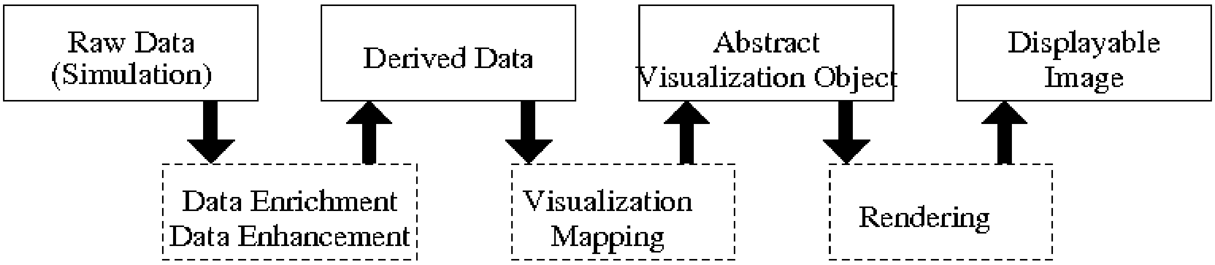


Figure 3.7.1: Conceptual visualization model.

Within the first step the raw data provided by the simulation is filtered or enhanced to extract the part the user is interested in. The following visualization mapping stage decides which type of visualization will occur by selecting a visualization algorithm like for example isosurface extraction. After the Abstract Visualization Object representing the visualization algorithm has been selected the rendering part produces the final image.

This general model of the visualization pipeline can be used to design distributed and parallel visualization scenarios. For example each stage of the pipeline can be scheduled separately thus offering more possibilities for exploiting different types of machines available on the grid. This especially refers to the execution of parallel visualization algorithms which have to be selected based on the hardware resources available.

Two types of parallelism are applicable: Task Parallelism and Data Parallelism. Task Parallelism refers to the pipeline structure of the visualization which can be sensibly exploited for speeding up the visualization process in case of multiple datasets that have to be visualized consecutively, such as a timeseries. Data Parallelism refers to parallelization within the pipeline stages by distributing the data among multiple threads or processes.

If a visualization is executed on the grid in a distributed and parallel manner it is commonly sensible to run the visualization job nearby the data to be visualized to save transmission time. In such a scenario the images as the final visualization output need to be transmitted to the local machine of the user. However the bandwidth requirements are significant if uncompressed images are transmitted at a high framerate. A sensible solution is therefore using a video codec to transmit the visualization output to the client without much loss of visual quality while offering significant bandwidth savings.

Another problem which becomes evident when a visualization is being executed remotely on the grid is the requirement for interaction. The users commonly want to steer the visualization or investigate different parts of the data based on the insights which they are gaining while examining the data. In general grid middleware is designed for batch processing, meaning that a job is submitted to a grid resource is running there until it finishes and the results are investigated in a post-mortem mode. Consequently interactivity has to be provided by additional tools such as glogin.

3.7.2. Glogin

Glogin is a tool for interactivity support on the grid, which can be applied for a wide variety of applications. It offers an interactive bi-directional channel between the grid and the users local desktop machine which can be applied for transmitting arbitrary data in and out of the grid.

The most important feature of glogin is its ability to interact with a grid application in a way similar to an application running on the local machine. This ability can be exploited for a lot of interactive applications since glogin itself can handle arbitrary network traffic.

Another distinctive feature is given by its light-weightness compared to deamon processes commonly offering interactive access services such as the ssh daemon.

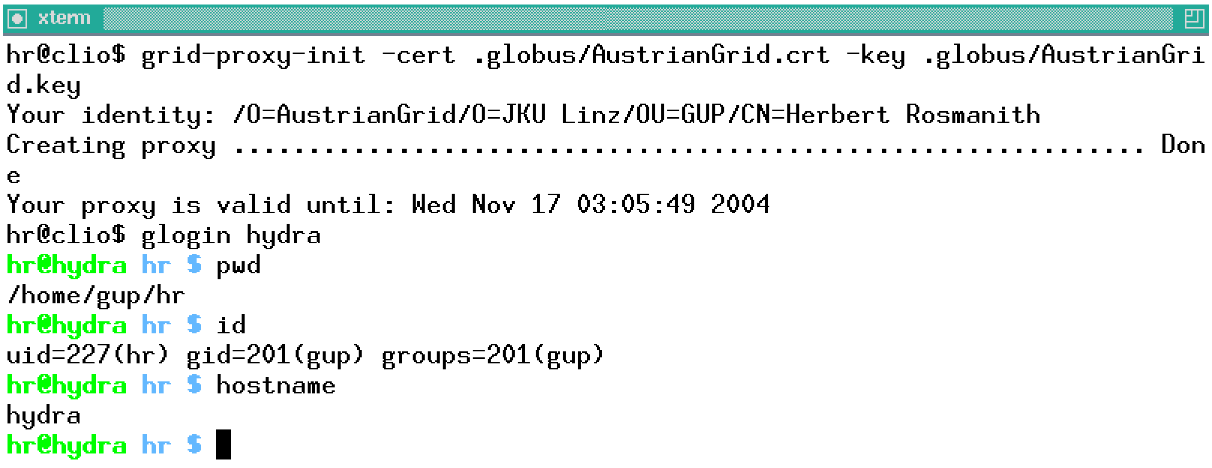
Due to its generic nature glogin offers a lot of possibilities to the users. The functionality is generally comparable to SSH and includes features such as interactive shell access to grid resources, TCP port forwarding, X11 forwarding, encryption, Virtual Private Networks, and transmission of generic low-level traffic, for example visualization data.

Shell Access. The most common usecase of glogin is the interactive shell access to remote grid resources. It is also the mode of operation where glogin is most "visible" to its users. It is used in way similar to ssh by invoking it from the command line:

```
hr@clio:~> glogin hydra
```

By using glogin in such a way, which is also shown in figure 3.7.2, one can log onto grid resources similar to using SSH while not needing a dedicated daemon being running on the server side. Another advantage is the fact that the user is identified by his grid certificate and thus is not required to enter a password upon login.

When invoked, glogin submits itself to the remote resource where it acts as a server instance. For transmitting the data of the shell session the GSH protocol is used, whose name was chosen because of the similarity to SSH.



```
hr@clio$ grid-proxy-init -cert .globus/AustrianGrid.crt -key .globus/AustrianGri
d.key
Your identity: /O=AustrianGrid/O=JKU Linz/OU=GUP/CN=Herbert Rosmanith
Creating proxy .............................................................. Don
e
Your proxy is valid until: Wed Nov 17 03:05:49 2004
hr@clio$ glogin hydra
hr@hydra hr $ pwd
/home/gup/hr
hr@hydra hr $ id
uid=227(hr) gid=201(gup) groups=201(gup)
hr@hydra hr $ hostname
hydra
hr@hydra hr $ █
```

Figure 3.7.2: Glogin shell access.

TCP Port Forwarding. Using this functionality glogin enables access to grid worker nodes with private IP addresses. The port forwarding technique applied is comparable to the one provided by SSH. If glogin is used to access a worker node with a private IP another instance of glogin is executed on another node within the same private network which offers a public IP address. This glogin process forwards the network traffic to the node with the private IP address in a manner transparent to the application using the connection.

Besides accessing nodes with private IP addresses the port forwarding functionality of glogin can also be used for securing the network traffic of another protocol if the port forwarding functionality is used in combination with the communication encryption functionality of glogin.

X11 Forwarding. The X11 forwarding feature is comparable to port forwarding. In a Unix/Linux environment (what is commonly used for grid installations) graphics requests generated by local and remote applications are handled by an X server. This X server output can be forwarded to a remote machine using "X11 forwarding". Glogin enables this type of connectivity for nodes which are accessible over the grid.

This functionality is an important feature to enable the transmission of graphics output from grid nodes to the local desktop, where the user is able to view the output of an interactive grid job.

If somebody is only considering ordinary batch jobs, the job output is commonly analysed post mortem. But if a user is running an interactive job on the grid the forwarding of X11 traffic is an important feature.

Encryption. Security has been an important consideration throughout the development of glogin. Therefore arbitrary traffic sent over glogin can be encrypted without further problems. The generic security service (GSS) is applied for encryption of glogin traffic.

If one combines this functionality with port forwarding arbitrary grid traffic can be encrypted seamlessly.

Virtual Private Networks. Glogin can also be applied for the realization of virtual private networks (VPNs). Compared to the approaches mentioned above the VPN functionality offers the possibility of routing additional types of network traffic such as UDP or ICMP packages over a secured network.

Low-Level Visualization Traffic. Applying glogin as a transport layer for visualisation data is an important usecase which has been applied within the scope of the CrossGrid project as well as the current int.eu.grid project.

However, from the point of view of glogin the transmission of visualization data is just an application, which makes use of the interactive capabilities of glogin. But since interactive visualization scenarios are very demanding in terms of network communication, they provide a good testcase for the capabilities of glogin.

Glogin Architecture. From the point of view of the middleware glogin is a simple grid job, which gets submitted by the resource management system. The requirement for an interactive bi-directional channel is common for all usecases of glogin. However, to date the grid middleware is the limiting factor for the creation of interactive bi-directional connections. This is due to the Globus GASS cache, which buffers the output of a grid job and which is only flushed each ten seconds or when the job terminates. Glogin circumvents this problem by applying the method outlined in figure 3.7.3.

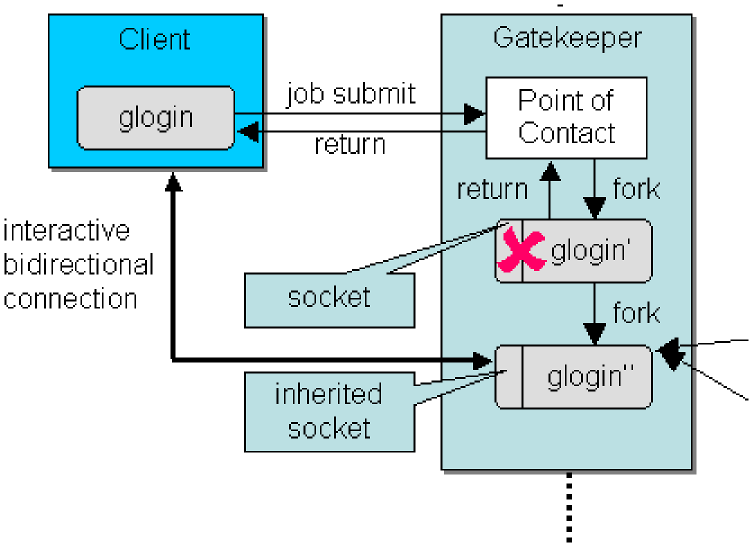


Figure 3.7.3: Glogin operation.

At first the client instance of glogin submits itself as a grid job to the remote resource, which the user wants to access. Upon startup the remote instance of glogin opens a connection endpoint for the client instance. In order to transmit the connection information to the remote client side it is written out and the server process forks itself. Now one instance terminates itself which leads to a flushing of the GASS cache, which subsequently enables the transmission of the connection information to the client side.

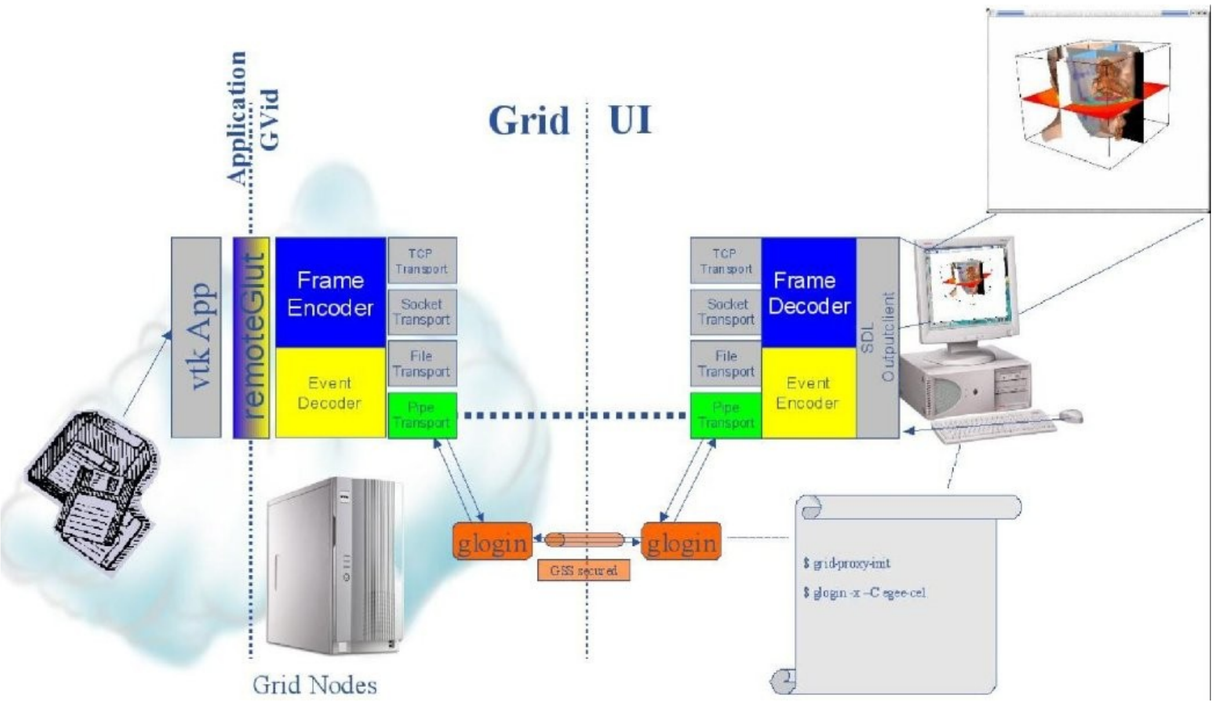
Upon receiving the information the client process can connect to the still running instance of the server process and thus generates an interactive bi-directional connection which can then be used for transmitting arbitrary traffic.

3.7.3. GVid

GVid is a grid-enabled video service, which is specifically useful in the context of grid-based visualizations. By using the GVid video streaming service a visualization can be rendered on the grid, while the user gets the visualization output delivered to his local desktop machine.

By using GVid the visualization itself can be executed on a remote grid resource, which enables the usage of powerful grid resources for visualization tasks. This scenario is especially sensible when a large amount of data needs to be visualized, meaning that the visualization is becoming a sensible grid job of its own right.

The functionality of GVid addresses two issues: The transmission of the visualization output to the users desktop but also the communication of the interaction events back to the remote rendering part running on the grid thus enabling a fully interactive remote visualization scenario. In general the GVid protocol can be realized over several different bi-directional communication channels. In the grid context, glogin offers itself as an ideal communication media since it exactly provides the bi-directional communication channel required. Within figure 3.7.4 the overall structure of GVid communicating over glogin is shown.



```
$ grid-proxy-init
$ glogin -x –C egee-cel
```

Figure 3.7.4: GVid structure.

GVid Architecture. For the communication of the visualization output from the rendering grid resource to the users machine the xvid video codec is used, which is commonly combined with an offscreen rendering library such as Mesa which enables graphics rendering on grid nodes without graphics hardware capabilities. On the clients side an interactive video player is required, which receives the video stream,decodes it and presents it to the user.

For enabling full interactivity for the remote visualization scenario user input events have to be sent back from the users local machine to the visualization running on the grid. This is done by using the GVid protocol, which contains built-in support for common mouse and keyboard interaction events. Within the graphics application these events can be applied for handling basic interaction modes like rotation and zooming.

For cases where more sophisticated user interaction is required such as interaction with a GUI the GVid protocol offers the possibility of user events which can encode and transmit arbitrary application data over the grid.

GVid offers several different input adapters for capturing the video stream as well as output adapters which are used to show the visualization to the user. Concerning the transmission protocol TCP socket communication or file transport can be used besides glogin.

One of the input adapters available enables grabbing its input directly from the X-server which is a rather generic approach. It offers the advantage that the visualization application does not need to be modified, while the efficiency of this approach is limited.

Another type of input adapter uses the RemoteGLUT library for offscreen rendering of an OpenGL application. Using this approach it does not require any graphics hardware on the grid worker nodes doing the rendering. Regarding the GVid output adapters two different clients have been developed, which are capable of displaying the incoming videostream and sending back the user input to the grid: One implementation is based on the SDL library, while the other one is a Java implementation.

3.7.4. An Example for Grid-based Remote Visualization

Within the int.eu.grid project a good example for an highly interactive grid-based visualization application was developed: A plasma fusion simulation, which calculates the particle trajectories within a fusion reactor. The simulation itself is done in parallel using MPI, while the master process of the parallel application also visualizes the simulation results using the OpenGL Graphics API. The rendering is executed as offscreen rendering by the master process which is subsequently using GVid to encode the visualization output and transmit it over the grid to the user. The transmission of the video stream is done over a glogin connection.

Using glogin and GVid for transmitting the visualization of the int.eu.grid fusion application enables running the parallel simulation of the fusion process on the grid testbed while the user can investigate and steer the simulation from his desktop machine. The interaction possibilities include full mouse event support including translation, rotation, and zooming of the scene displayed.

By submitting the fusion application together with glogin using the Migrating Desktop (MD) [KUPMIG] as user interface, glogin acts as communication channel between the parallel fusion application and the Roaming Access Server (RAS), which acts as interface between the MD and the grid middleware. The second part of the interactive connection is provided by a https stream interconnecting the RAS server and the MD running on the users machine. The whole scenario is illustrated in figure 3.7.5.

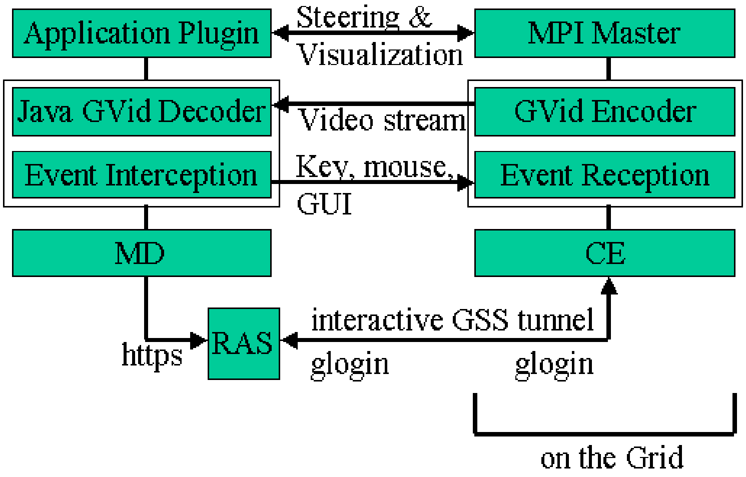


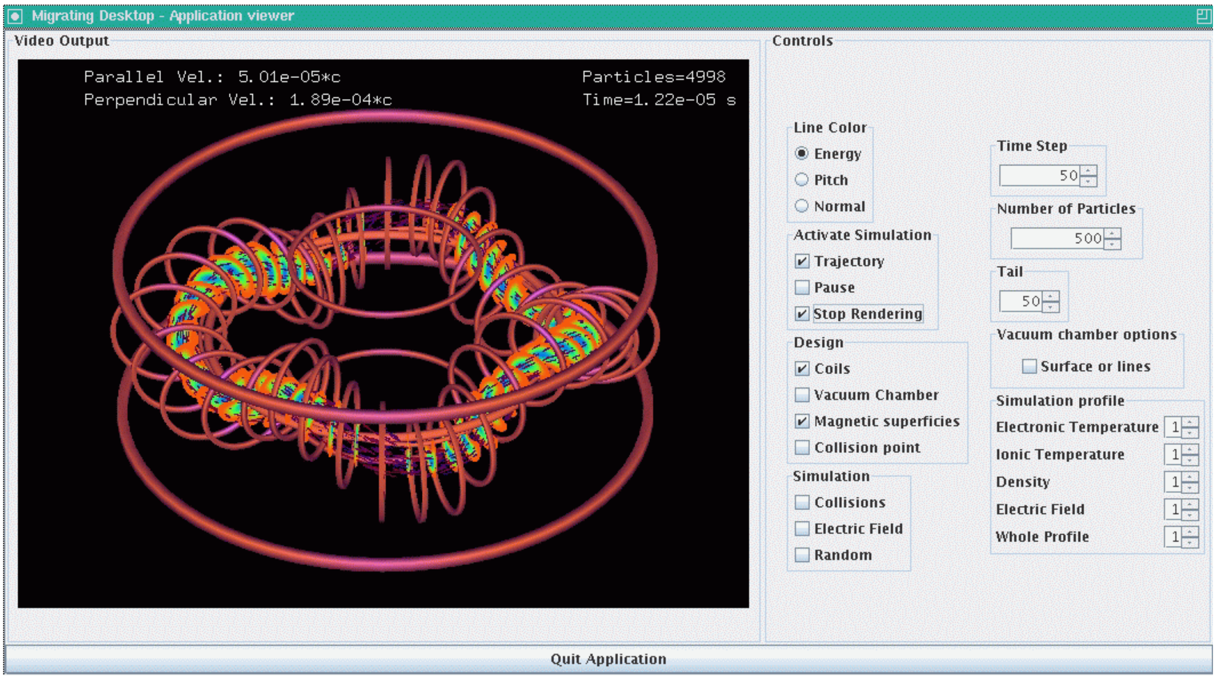
Figure 3.7.5: Interactive visualization architecture.

The communication channel provided by glogin is interconnecting the user the master process of the OpenMPI-based fusion application, which is producing the OpenGL-based rendering.

The Java-based user interface provides two general types of interaction: On the one hand interaction with the rendering part of the fusion application enabling the usage of the mouse for rotating, translating and zooming the model of the fusion reactor shown, on the other hand a panel containing forms for setting simulation and visualization parameters, which are also directly communicated to the fusion application on the grid. If the parameter change only affects the rendering the event is processed by the master node, while changes affecting the simulation are propagated to the simulation processes using MPI communication routines.

While the mouse events are propagated within the standard GVid protocol, the transmission protocol of GVid has been extended to support the transmission of the user interface events. This extension enables the GVid protocol to transmit arbitrary event data, which enables the remote steering of arbitrary applications.

Figure 3.7.6 shows the visualization output produced by the fusion application and transmitted over glogin together with the user interface plugin used to steer the remote simulation.



Migrating Desktop - Application viewer

Video Output

```
Parallel Vel.: 5.01e-05*c                    Particles=4998
Perpendicular Vel.: 1.89e-04*c               Time=1.22e-05 s
```

Controls

Line Color
● Energy
○ Pitch
○ Normal

Activate Simulation
☑ Trajectory
□ Pause
☑ Stop Rendering

Design
☑ Coils
□ Vacuum Chamber
☑ Magnetic superficies
□ Collision point

Simulation
□ Collisions
□ Electric Field
□ Random

Time Step: 50

Number of Particles: 500

Tail: 50

Vacuum chamber options
□ Surface or lines

Simulation profile
Electronic Temperature: 1
Ionic Temperature: 1
Density: 1
Electric Field: 1
Whole Profile: 1

Figure 3.7.6: Fusion application in MD.

3.7.5. References

[AFSVLA] H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.G. van den Brand, G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D. Vasunin, A. Visser, H.H. Yakali: VLAM-G: A Grid-Based Virtual Laboratory, Scientific Programming Journal, Vol. 10, No. 2, pp. 173-181, 2002

[BROAPP] J. Brooke, T. Eickermann, U. Woessner: Application Steering in a Collaborative Environment, In Proc of the 2003 IEEE/ACM Conference on Supercomputing, Phoenix, AZ, USA, November 2003

[CHIACC] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, T. Udeshi: Access Grid: Immersive Group-to-Group Collaborative Visualization, In Proc. of the 4th International Immersive Projection Technology Workshop, Ames, IA, USA, June 2000

[EICSTE] T. Eickermann, W. Frings, P. Gibbon, L. Kirtchakova, D. Mallmann, A. Visser: Steering UNICORE Applications with VISIT, Philosophical Transactions: Mathematical, Physical and Engineering Sciences, Vol. 363, No. 1833, The Royal Society, London, UK, pp. 1855-1865, August 2005

[JANDEP] T.J. Jankun-Kelly, O. Kreylos, K. Ma, B. Hamann, K.I. Joy, J. Shalf, E.W. Bethel: Deploying Web-Based Visual Exploration Tools on the Grid, IEEE Computer Graphics and Applications, Vol. 23, No. 2, pp. 40-50, March/April 2003

[FOSGLO] I. Foster, C. Kesselman: Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputing Applications, Vol. 11, No. 2, pp. 115-128, 1997

[SUZVIS] Y. Suzuki, K. Sai, N. Matsumoto, O. Hazama: Visualization Systems on the Information-Technology-Based Laboratory, IEEE Computer Graphics and Applications, Vol. 23, No. 2, pp. 32-39, March/April 2003

[UPSTHE] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam: The Application Visualization System: A Computational Environment for Scientific Visualization, Computer Graphics and Applications, Vol. 9, No. 4, pp. 30-42, July 1989

[BETGRI] E.W. Bethel, J. Shalf: Grid-Distributed Visualizations Using Connectionless Protocols, IEEE Computer Graphics and Applications, Vol. 23, No. 2, pp. 51-59, March/April 2003

[ALLTHE] G. Allen, W. Benger, T. Goodale, H. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidl, J. Shalf: The Cactus Code: A Problem Solving Environment for the Grid, in Proc. of the 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, USA, pp. 253-262, 2000

[NORENA] A. Norton, A. Rockwood: Enabling View-Dependent Progressive Volume Visualization on the Grid, IEEE Computer Graphics and Applications, Vol. 23, No. 2, pp. 22-31, March/April 2003

[AESPRE] M. Aeschlimann, P. Dinda, L. Kallivokas, J. Lopez, B. Lowekamp, D. O'Hallaron: Preliminary Report on the Design of a Framework for Distributed Visualization, In Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, NV, USA, pp. 1833-1839, June 1999

[LOWDES] B. Lowekamp, N. Miller, R. Karrer, T. Gross, P. Steenkiste: Design, Implementation, and Evaluation of the Remos Network Monitoring System, Journal of Grid Computing, Vol. 1, No. 1, pp. 75–93, June 2003

[WOLTHE] R. Wolski, N. Spring, J. Hayes: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757–768, October 1999

[SCHTHE] W.J. Schroeder, K.M. Martin, W.E. Lorensen: The Visualization Toolkit. An Object Oriented Approach to 3D Graphics, Prentice Hall, 1996

[FOSDIS] I. Foster, J. Insley, C. Kesselman, G. von Laszewski, M. Thiebaux: Distance Visualization: Data Exploration on the Grid, IEEE Computer, Vol. 32, No. 12, pp. 36-43, December 1999

[CZAPRA] K. Czajkowski, A.K. Demir, C. Kesselman, M. Thiebaux: Practical Resource Management for Grid-based Visual Exploration, In Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing, San Francisco, CA, USA, pp. 416-423, August 2001

[KUPMIG] M. Kupczyk, R. Lichwała, N. Meyer, B. Palak, M. Płóciennik, P. Wolniewicz: Migrating Desktop Interface for Several Grid Infrastructures, In Proc. of the Parallel and Distributed Computing and Networks Conference PDCN 2004, Innsbruck, Austria, February 2004

[CROPAR] T.W. Crockett: Parallel Rendering, Tech. Rep. TR-95-31, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA, USA, April 1995

[HABVIS] R.B. Haber, D.A. McNabb: Visualization Idioms: A Conceptual Model for Scientific Visualization Systems, In: G.M. Nielson, B. Shriver, L.J. Rosenblum (Eds.): Visualization in Scientific Computing, IEEE Computer Society Press, ISBN 0-8186-5979-3, pp. 74–93, 1990

3.8. gLite

3.8.1. The gLite Architecture

The Enabling Grids for E-sciencE project (EGEE), a European flagship research infrastructures grid project and the world's largest grid infrastructure of its kind (it involves more than 70 partners from 27 countries, arranged in twelve regional federations, and provides more than 20000 CPUs, almost 200 sites and 10 petabytes of available network storage), deploys the gLite middleware [3.8_18], a middleware stack that combines components developed in various related projects, in particular Condor [3.8_7], Globus [3.8_12], LCG [3.8_19], and VDT [3.8_29], extended by EGEE developed services. This middleware provides the user with high level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the grid infrastructure as well as grid applications, all embedded into a consistent security framework.

The gLite grid services follow a Service Oriented Architecture [3.8_25], which will facilitate interoperability among grid services and allow easier compliance with upcoming standards, such as OGSA, that are also based on these principles. The services are expected to work together in a concerted way in order to achieve the goals of the end-user; however, they can also be deployed and used independently, allowing their exploitation in different contexts.

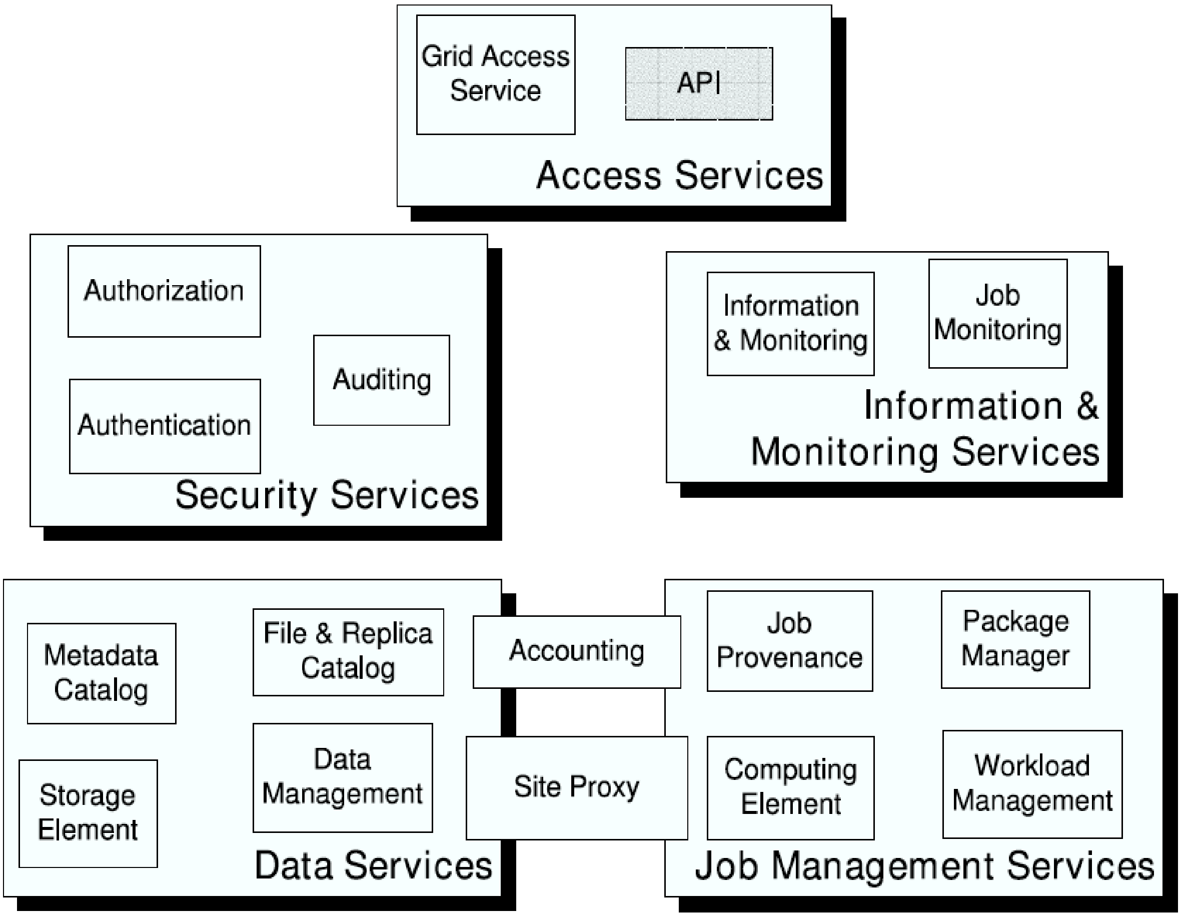


Figure 3.8.1: gLite services.

Figure 3.8.1 depicts the high level services, which can thematically be grouped into 5 service groups: Security services encompass the Authentication, Authorization, and Auditing services which enable the identification of entities (users, systems, and services), allow or deny access to services and resources, and provide information for post-mortem analysis of security related events. It also provides functionality for data confidentiality and a dynamic connectivity service, i.e. a means for a site to control network access patterns of applications and grid services utilizing its resources. Information and Monitoring Services provide a mechanism to publish and consume information and to use it for monitoring purposes. The information and monitoring system can be used directly to publish, for example, information concerning the resources on the grid. More specialized services, such as the Job Monitoring Service and Network Performance Monitoring services, can be built on top.

Job Management Services. The main services related to job management/execution are the computing element, the workload management, accounting, job provenance, and package manager services. Although primarily related to the job management services, accounting is a special case as it will eventually take into account not only computing, but also storage and network resources. The Computing Element (CE) provides the virtualization of a computing resource (typically a batch queue of a cluster but also supercomputers or even single workstations). It provides information about the underlying resource and offers a common interface to submit and manage jobs on the resource. The Workload Management System (WMS) is a grid level meta-scheduler that schedules jobs on the available CEs according to user preferences and several policies. It also keeps track of the jobs it manages in a consistent way via the logging and bookkeeping service. The Job Provenance (JP) service provides persistent information on jobs executed on the grid infrastructure for later inspections, data-mining operations, and possible re-runs. Finally, the Package Manager (PM) service allows the dynamic deployment of application software. While the CE and WMS are part of the production gLite 3.0 release, the JP and PM are only available as prototypes.

Data Services. The three main services that relate to data and file access are:

- Storage Element
- file and replica catalog services
- data management

In all of the data management services described below the granularity of the data is on the file level. However, the services are generic enough to be extended to other levels of granularity. The Storage Element (SE) provides the virtualization of a storage resource (which can reach from simple disk servers to complex hierarchical tape storage systems) much as the CE does for computational resources. The catalog services keep track of the data location as well as relevant metadata (e.g. checksums and file-sizes) and the data movement services allow for efficient managed data transfers between SEs. The access to files is controlled by Access Control Lists (ACLs). Application specific metadata is expected not to be stored in the basic gLite services but in application specific metadata catalogs. All the data management services act on single files or collections of files. To the user of the EGEE data services the abstraction that is being presented is that of a global file system. A client user application may look like a Unix shell, which can seamlessly navigate this virtual file system, listing files, changing directories, etc. Note that the gLite architecture does not in general impose specific deployment scenarios (i.e. how many instances of a certain service are available to a user, if a service is replicated or distributed, etc.).

A Virtual Organisation (VO) comprises a set of individuals and/or institutions having access to computers, software, data, and other resources for collaborative problem-solving or other purposes. Virtual Organisations are a concept that supplies a context for operation of the grid that can be used to associate users, their requests, and a set of resources. Most importantly, service instances may serve multiple VOs will facilitate the scalability and performance of the grid system although a VO may require its own instance as well.

3.8.2. Security

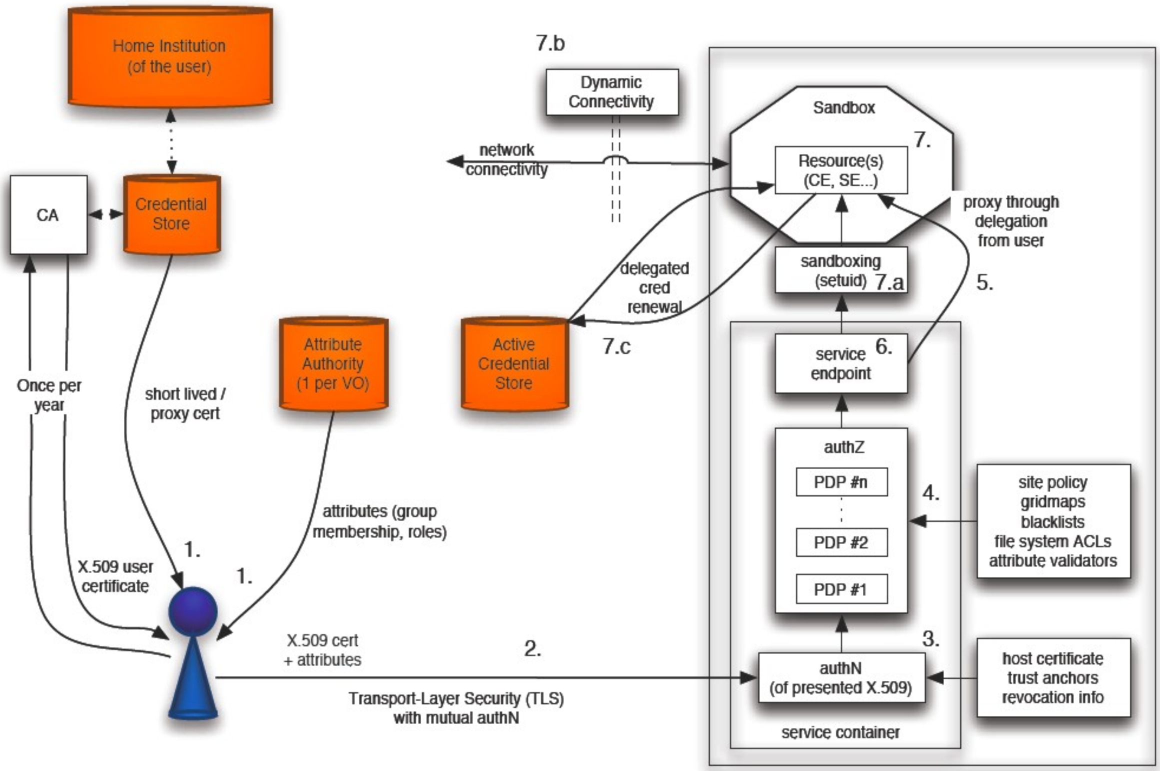


Figure 3.8.2: Security architecture components, and a user (agent) accessing a resource.

Figure 3.8.2 depicts an overview on how the components in the security architecture interact in the following typical request flow:

1. The user obtains grid credentials from a credential store (2), and the necessary tokens that assert the user's rights to access the resource (3). The credentials are short-lived and often derived from longer-term credentials, such as X.509 identity certificates issued by a Certification Authority (CA). EGEE uses myProxy [3.8_16] as credential store and the Virtual Organization Membership Service VOMS [3.8_22] as attribute authority. VOMS is also used to manage the membership of VOs.
2. The user and the service container authenticate identities to each other and establish a secure communication channel across the (open) network with integrity, authenticity and confidentiality protection, and over which a SOAP message payload is conveyed. By default, this is accomplished by use of HTTP over TLS. The established connection event is logged.
3. During the authentication in step 2 the authentication layer validates the user's identity with the trust anchors and credential revocation information, if such exists. The result of the validation is logged. The service container absorbs the payload and routes it to the correct service endpoint. In the case of message-level security, the authentication and integrity checks happen here (i.e., after the message has been absorbed from the network).

4. The authorization routines ensure that the user has permission to access the resource, by combining attribute assertions and the VO policy (sent with the request) with the local site policy and other sources of access control.
5. In the case that delegated credentials are used, the user delegates rights to the delegating resource to act on the user's behalf. Note however that delegation typically happens as a separate end-point invocation, and is part of the application-level message flow between the user and the service.
6. The service implementation gets invoked. The authorization routines may be used for additional evaluation and consultation.
7. The service interacts with the resource, which in turn may have delegated credentials at its disposal.

Sandboxing and isolation techniques limit the user's influence on the resource to within the expected boundaries, avoiding malicious or unintended usage or in the worst scenario a security breach. These include:

- operating the resource in a different user space than that of the service container
- consulting the Dynamic Connectivity service in order to temporarily enable direct inbound and/or outbound network connectivity to the resource
- providing additional protection of the delegated credentials by use of an active credential store

This is also useful in the case of long-term use of a resource, where a renewal of the delegated credentials may be necessary.

3.8.3. Information and Monitoring Services

The gLite system for information and monitoring is R-GMA [3.8_23, 3.8_14], which is a Relational implementation of the Grid Monitoring Architecture [3.8_27] from the GGF [3.8_9]. R-GMA has been designed to be easy for end users to publish information (from a batch job or otherwise) and query that information in a grid environment.

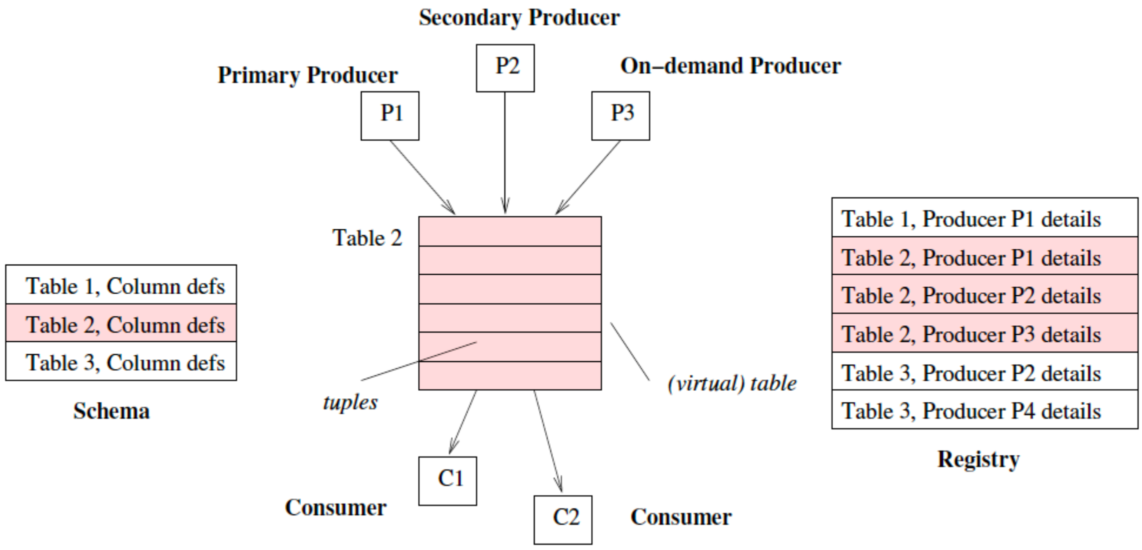


Figure 3.8.3: R-GMA components.

Figure 3.8.3 shows the principal components of R-GMA. Data is written into the R-GMA virtual database by producers and read from it by consumers. R-GMA is not a distributed database management system. Instead, it provides a useful and predictable information system built on a much looser coupling of data providers across agrid.

Defining the schema. The first task for the user is to define what needs to be published. This has to be one or more tables following the relational model. A common technique in design of a relational schema is to make use of "surrogate keys": a small integer, which can be used as a foreign key to establish a relationship. A traditional case would be to assign an departmentId to each department and then to include this as a column of the employee table. This works well for a single managed database with a mechanism to assign departmentId values, but it does not work in the grid. You should not assume anything about what anybody else is publishing. It is best to think of publishing a series of measurements of the same quantity but made at different times; all R-GMA tuples (records) have an associated timestamp and the R-GMA query types take advantage of this.

Producers are the data providers for the virtual database. Writing data into the virtual database is known as "publishing", and data is always published in complete rows, known as tuples. There are three classes of producer: Primary, secondary and on-demand. Each is created by a user application and returns tuples in response to queries from consumers. The main difference is in where the tuples come from. There are three ways considered here to for a job to publish data into R-GMA. The least intrusive is to use a job wrapper, which can publish information on the state of the job picked up by looking at stdout. This can be done without any modifications to the job itself, provided that useful information can be gleaned from stdout. The job wrapper will insert data into the R-GMA system by means of a primary producer which will have four important R-GMA calls:

- create primary producer with appropriate properties
- declare table with predicate: this information goes into the registry
- insert tuples into virtual database
- close primary producer

A second alternative is to insert R-GMA calls directly into the application code. This might be done using any of the supported APIs: C, C++, Java and Python. The code, from an R-GMA viewpoint, is identical to that used above in the job wrapper. A third approach is to use the native logging API (e.g. log4cxx or log4j). You will then need to run the program with an R-GMA appender, which is provided for Java and C++. This takes the messages, which might otherwise have gone to the terminal or to syslog and sends them to an R-GMA producer. This is an attractive solution in that it requires that the user can just use his existing logging mechanisms but has the disadvantage that it it is not possible to modify the schema. You may wish to collect information together into a secondary producer, which is capable of answering latest or history queries. If so you should probably set up two of them for some redundancy. For the sake of this example we will assume that you wish to store history so you create a secondary producer to answer history queries.

Consumers. In R-GMA, each consumer represents a single SQL SELECT query on the virtual database. The query is first matched against the list of available producers in the registry and a set of producers capable of answering the query is selected. There are four query types: continuous, latest, history and static. They are all expressed by a normal SQL query though there are some restrictions on the continuous query as this simply acts as a filter on published tuples and so joins and aggregate functions are not permitted. If you issue a continuous query you will receive every tuple satisfying the query as it is published. Such a query has no natural end. The latest query only considers those tuples, which were most recently published. Tables have a primary key defined to allow latest tuples to be defined. You can then query the information — if you perform a continuous query you will be connected to the primary producers but if you carry out a history query you will be connected to the secondary producer, which was created to answer history queries.

Command Line Tool. An easy to use command line tool (written in Python) is also provided with a built-in help system. This tool accepts short commands and provides defaults for as much as possible. For example:

```
rgma> SELECT Name, Endpoint FROM Service
```

where rgma> is the prompt, will issue a query using the current values of parameters such as the type of query, the timeout etc. The current values can be changed or displayed:

```
rgma> SET QUERY CONTINUOUS
rgma> SET TIMEOUT 3 minutes
rgma> SHOW MAXAGE
```

Command history and command completion are also provided.

Service Discovery. The approach taken to service discovery was an API hiding the underlying information system. The information system is linked in via a plug-in mechanism for which we currently support R-GMA, bdII and an XML file. APIs are provided in C and Java and allow a user (or another service) to select a suitable service.

To understand more of how to use R-GMA for monitoring and of how to use the Service Discovery APIs please consult the R-GMA documentation [3.8_14].

3.8.4. Workload Management Services

The Workload Management System (WMS) comprises a set of grid middleware components responsible for the distribution and management of tasks across grid resources, in such a way that applications are efficiently executed. The specific kind of tasks that request computation are usually referred to as "jobs". In the WMS, the scope of tasks needs to be broadened to take into account other kinds of resources, such as storage or network capacity. This change of definition is mainly due to the move from batch-like activity to applications with more demanding requirements for data access or interactivity, both with the user and with other tasks. The core component of the Workload Management System is the Workload Manager (WM), whose purpose is to accept and satisfy requests for job management coming from its clients. The other fundamental component is the Job Logging and Bookkeeping Service, which is described below. For a computational job there are two main types of request: submission and cancellation. The status request is managed by the Logging and Bookkeeping Service. In particular the meaning of the submission request is to pass the responsibility of the job to the WM. The WM will then pass the job to an appropriate CE for execution, taking into account the requirements and the preferences expressed in the job description. The decision of which resources should be used is the outcome of a matchmaking process between submission requests and available resources. The availability of resources for a particular task depends not only on their state, but also on the utilization policies that the resource administrators and/or the administrator of the VO the user belongs to have put in place.

gLite Job Description Language. A job passed to the gLite WMS needs to be described in a specific language, the gLite Job Description Language (JDL). The JDL used for gLite, and originally developed for the EU DataGrid project, is based on the Condor ClassAd language [3.8_24]. Its central construct is a record-like structure, the classad, composed of a finite number of distinct attribute names mapped to expressions. An expression contains literals and attribute references composed with operators in a C/C++ like syntax. These ads conform to a protocol that states that every description should include expressions named Requirements and Rank, which denote the requirements and preferences of the advertising entity. Two entity descriptions match if each ad has an attribute, Requirements, that evaluates to true in the context of the other ad. The main advantages of this framework can be summarized by the following three points:

- it uses a semi-structured data model, so no specific schema is required for the resources description, allowing it to work naturally in a heterogeneous environment
- the language folds the query language into the data model. Requirements (i.e. queries) may be
- expressed as attributes of the job description
- ClassAds are first-class objects in the model, hence descriptions can be arbitrarily nested, leading to a natural language for expressing resources and jobs aggregates (e.g. DAGs) or co-allocation requests

The gLite JDL defines specific attributes to specify:

- batch or interactive, simple, MPI-based, checkpointable and partitionable jobs;
- aggregates of jobs with dependencies (Directed Acyclic Graphs);
- constraints to be satisfied by the selected computing and storage resources;
- data access requirements: appropriate conventions have been established to express constraints about the data that a job wants to process together with their physical/logical location within the grid;
- preferences for choosing among suitable resources (ranking expressions)

As mentioned, the JDL is semi-structured and extensible. A set of predefined attributes have a special meaning for the underlying components of the Workload Management System. Some of them are mandatory, while others are optional. The set of predefined attributes [3.8_13] can be decomposed in the following groups:

- Job attributes: representing job specific information and specifying actions that have to be performed by the WMS to schedule the job
- Data attributes: representing the job input data and Storage Element related information. They are used for selecting the resources from which the application has the best access to data
- Requirements and Rank: allowing the user to specify respectively which are the needs and preferences, in term of resources, of their applications.

The Requirements and Rank expressions are built using the Resources Attributes, which represent the characteristics and status of the resources and are recognizable in the job description as they are prefixed with the string "other". The Resources attributes are not part of the predefined set of attributes for the JDL as their naming and meaning depends on the adopted Information Service schema [3.8_10] for publishing such information. This independence of the JDL from the resources information schema allows targeting for the submission resources that are described by different Information Services without any changes in the job description language itself.

WMS User Interfaces. After having created the descriptions of their applications, users expect to be able to ignore the complexity of the grid resources and to be enabled to submit them to the Workload Management System and monitor their evolution over the grid.

The functionalities the WMS provides include the following:

- Job (including DAGs) submission for execution on a remote Computing Element, also including:

  - automatic resource discovery and selection,
  - staging of the application input sandbox,
  - restart of the job from a previously saved checkpoint state,
  - interactive communication with the running job,
- Listing of resources suitable to run a specific job according to job requirements,
- Cancellation of one or more submitted jobs,
- Retrieval of the output files of one or more completed jobs,
- Retrieval of the checkpoint state of a completed job,
- Retrieval of jobs bookkeeping and logging information.

All this functionality is made available through a command line interface and an API providing C++ and Java bindings. GUI components have been developed on top of the Java API.

The WMS client API supplies the client applications with a set of interfaces over the job submission and control services made available by the gLite WMS through a web service based interface. The API provides the corresponding method for each operation published in the WSDL description of the WMProxy Service (http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/wmproxy). The request types supported by the WMProxy Service are:

- **job:** a simple application
- **DAG:** a directed acyclic graph of dependent jobs
- **collection:** a set of independent jobs

Jobs in turn can be batch, interactive, MPI-based, checkpointable, partitionable and parametric. The specification of the JDL for describing the request types is available at [3.8_13]. Besides requests submission, the WMProxy also exposes additional functionality for request management and control such as cancellation, job files perusal and output retrieval. Requests status follow-up can be instead achieved through the functionality exposed by the Logging & Bookkeeping (LB) service [3.8_17]. The documentation describing the WMProxy Client API providing C++, Java and Python bindings can be found at [3.8_32].

The Logging and Bookkeeping service (L&B) [3.8_17, 3] is used by WMS internally to gather different information on running jobs and provide the user with an overall view on the job state. The service collects events in a non-blocking asynchronous way with a robust delivery mechanism. The job state is computed on the fly at the bookkeeping database, using a state machine that tolerates even out of order event delivery. Besides gathering the "system" information on running jobs, the service can also collect user information in the form of arbitrary "name = value" tags (annotations) assigned to a job, both from a running application or independently. The job status information gathered by the LB is made available through the gLite user-interface commands. In addition to this simple querying mechanism, the user can pose simple or more complex queries with the public L&B API (available in C and C++ or as a web-service interface). Examples of such queries are:

- state of a concrete job,
- details on all user's running jobs,
- jobs that are running on a concrete computing element,
- user's jobs that returned exit code between e.g. 2 and 7,
- user's jobs resubmitted in last two hours,

- user's jobs, annotated as green or red color, that started execution in the first week of January,
- user's failed jobs that were marked as red first, and then re-colored to green,
- red-colored jobs, heading to a computing element at which the user's job have recently failed.

The list of more or less random examples presented here demonstrates the strength of the L&B API. The user can also register for receiving notifications when a job enters a state matching conditions specified in a similar way. Job state information is also fed into the R-GMA infrastructure to provide yet another way of accessing the job bookkeeping information. More detailed examples of use of the LB service are discussed in detail in [3.8_17], including appropriate code fragments.

3.8.5. Data Management Services

Storage. gLite relies on storage systems exposing an SRM [3.8_11] interface. Current systems supported include Castor (http://cern.ch/castor), dCache (http://www.dcache.org/) and the gLite Disk Pool Manager (DPM). The DPM has been developed as a lightweight solution for disk storage management offering much of the functionality of dCache but avoiding its complexity. DPM is security enabled, providing ACL based authentication to file access. In addition to the SRM interface, DPM offers an rfio interface for posix like data access and GridFTP [3.8_6] for data transfer. This is also the mechanism the gLite file transfer service described below, is using. In order to shield the user from the differences the current storage systems expose in their posix-like access libraries, gLite provides a Grid File Access Library (GFAL), a C API posix-like interface that provides methods such as gfal open, gfal read, etc. GFAL interaces with the different SRM implementations (including their native posix access mechanisms) and GridFTP.

Catalogs. gLite provides a catalog, named LFC, to store the location(s) of their files and replicas. LFC will map LFNs or GUIDs to SURLs. It is a high performance file catalogue that builds on the experiences gathered from the EGEE user communities. The LFC supports Oracle and Mysql as database backends, and is integrated with the GFAL interface. It shares the codebase with the name service part of the DPM, discussed above. Similarly to the DPM, the LFC exposes methods to the user through the GFAL interface that, in turn, interacts with the SRM implementations and GridFTP. The LFC client has a POSIX-like command line interface with commands such as lfc-chmod,lfc-ls,lfc-rm.

Data Movement. The gLite File Transfer Service FTS is a low level data movement service, responsible for moving sets of files from one site to another while allowing participating sites to control the network resource usage. This control includes the enforcement of site and usages policies such as fair-share mechanisms on dedicated network links. It is designed for point to point movement of physical files. The FTS has dedicated interfaces for managing the network resource and to display statistics of ongoing transfers. The FTS is also able to communicate with external Grid File Catalogs, i.e. the file to be transferred can also be specified using an LFN. The FTS has three interfaces that can be used for programming. The File Transfer Interface is used to submit File Transfer jobs, get status on current jobs, list requests in a given job state, cancel transfers, set priority of transfers; and to add, remove and list VO managers. The Channel Management Interface can be used to add, list and delete channels for the FTS instance, and set channel parameters. It has also methods to add, remove and list channel managers and to apply policies for jobs that need manual intervention, such as being in HOLD state. Finally, the Status Interface can be used to list or summarize the channel and VO activity, and to list all running background Transfer Agent processes. There is a set of command line tools available that interact with these interfaces, performing these tasks by contacting the FTS. All the FTS interfaces come with WSDL descriptions and the user can actually use the WSDL to generate clients for any language needed. The gLite distribution includes a set of client APIs for Java, C/C++ and Perl. As a secure connection is used to talk to the FTS web service, a valid GSI proxy is necessary. The VOMS extensions are needed if the client wants to contact for example the Channel Management interface. VO and site managers, who should have an extra "admin" group membership signed by VOMS, should only use this. The FTS Transfer Interface's transferSubmit method takes as input a TransferJob object, which consists of:

- an array of TransferJobElements each describing an individual file transfer within the job (source and destination pairs),
- a list parameters (key, value pairs) for transfer layer specific paramaters that are applied to each file transfer (e.g. GridFTP parameters), and
- the credential that is used by the transfer system to retrieve the appropriate proxy for the transfer.

The rest of the FTS Transfer Interface, the Channel Management Interface and Status Interface methods are simple and straightforward setters and getters very much in Java style, that can easily be used like any other RPC call through SOAP. The detailed syntax for the API and all command line tools is described in the userguide [3.8_4].

3.8.6. References

[3.8_1] V. Breton et al. EGEE Deliverable 4.4: Second Revision of EGEEApplication Migration Progress Report. https://edms.cern.ch/document/707799/

[3.8_2] DICOM Digital Imaging and Communication in Medicine. http://dicom.nema.org/

[3.8_3] F. Dvorak et al. Services for Tracking and Archival of Grid Job Information. In Proceedings Cracow Grid Workshop'05, 2006. http://www.cyfronet.krakow.pl/cgw05

[3.8_4] EGEE JRA1. EGEE File Transfer Service User Guide. https://edms.cern.ch/document/591792/

[3.8_5] Karl Cajkowski et. al. The WS-Resource Framework, 2004. http://www-106.ibm.com/ developerworks/library/ws-resource/ws-wsrf.pdf

[3.8_6] W. Allcock et al. GridFTP Protocol Specification. Global Grid Forum Recommendation GFD.20, March 2003.

[3.8_7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, 5(3):237–246, 2002.

[3.8_8] glite command line interface reference, https://edms.cern.ch/document/674643/1

[3.8_9] Global Grid Forum. http://www.gridforum.org/

[3.8_10] Homepage of the GLUE Schema Activity. http://infnforge.cnaf.infn.it/glueinfomodel

[3.8_11] The GGF Grid Storage Resource Manager Working Group

[3.8_12] I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid. The International Journal ofHigh Performance Computing Applications, 15(3):200–222, Fall 2001

[3.8_13] JDL Attributes Specification. https://edms.cern.ch/document/590869/1.EGEE-JRA1-TEC-590869-JDL-Attributes-v0-4

[3.8_14] JRA1-UK. http://hepunx.rl.ac.uk/egee/jra1-uk/

[3.8_15] EGEE JRA3. EGEE Deliverable 3.3: Global Security Architecture (1st revision). https://edms.cern.ch/document/602183/

[3.8_16] D. Kouril and J. Basney. A Credential Renewal Service for Long-Running Jobs. In Grid 2005 — 6th IEEE/ACM International Workshop on Grid Computing, Seattle, US, November 2005

[3.8_17] A. Kˇrenek et al. L&B Users Guide. https://edms.cern.ch/file/571273/1/LB-guide.pdf

[3.8_18] E. Laure, F. Hemmer, et al. Middleware for the Next Generation Grid Infrastructure. In Computing in High Energy and Nuclear Physics (CHEP), Interlaken, Switzerland, September 2004.

[3.8_19] The LCG Project, http://cern.ch/lcg

[3.8_20] Paul J. Leach and Rich Salz. UUIDs and GUIDs, February 1998.

[3.8_21] TheWeb Services Interoperability Organization. WS-I Documents. http://www.ws-i.org/Documents.aspx

[3.8_22] R. Alfieri and others. VOMS, an Authorization System for Virtual Organizations. In Grid Computing, First European Across Grids Conference, 2004.

[3.8_23] R-GMA. http://www.r-gma.org/

[3.8_24] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7), Chicago, IL, July 1998.

[3.8_25] D. Sprott and L. Wilkes. Understanding Service-Oriented Architecture. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp

[3.8_26] EGEE Middleware Design Team. EGEE Deliverable 1.4: EGEE Middleware Architecture. https://edms.cern.ch/document/594698/.

[3.8_27] B. Tierney, R. Aydt, et al. A grid monitoring architecture. Technical Report GWD-Perf-16-1, GGF, 2001.

[3.8_28] S. Tuecke et al. Open Grid Services Infrastructure (OGSI) — Version 1.0. https://forge.gridforum. org/projects/ogsi-wg

[3.8_29] The Virtual Data Toolkit. http://www.cs.wisc.edu/vdt/

[3.8_30] GGF OGSA WG. Open Grid Services Architecture — Glossary of Terms. https://forge.gridforum.org/projects/ogsa-wg

[3.8_31] GGF OGSA WG. The Open Grid Services Architecture, Version 1.0. https://forge.gridforum.org/projects/ogsa-wg

[3.8_32] WMproxy API documentation. http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/ glite-wmproxy-api-index.shtml

4. Infrastructure

4.1. Virtual Laboratory

In general, the Virtual Laboratory is a distributed environment, providing a remote access to the various kind of scientific equipment and computational resources. The Virtual Laboratory (VLab) has been developed in Poznań Supercomputing and Networking Center since the beginning of the year 2002. Users can submit their tasks in form of Dynamic Measurement Scenario — the set of experiments and computational tasks of different kind. These tasks form a dependency graph describing the connections between them and all possible flow paths — the actual flow path is determined upon run-time — based on results obtained on each stage. The tasks are scheduled on one-by-one (or on group) basis, after the appropriate results from the preceding tasks are obtained. The Virtual Laboratory is not a standalone system. It was designed to cooperate with many other grid systems, providing only the purpose-specific functionality and relying on well known and reliable grid solutions. The most important system the VLab cooperates with is the Globus Toolkit, in the scope of scheduling computational tasks, software services and libraries for resource monitoring, discovery, and management. All computational tasks submitted in the VLab system are transferred to the Globus via the GRMS module — an important part of the GridLab project. Among other external systems used by the Virtual Laboratory are: VNC, a data management system, an authentication module and a RAD authorization system.

4.1.1. The Virtual Laboratory Architecture

The Virtual Laboratory system has a modular architecture. The modules can be grouped into three main layers. The general diagram is presented below in the figure 4.1.1.

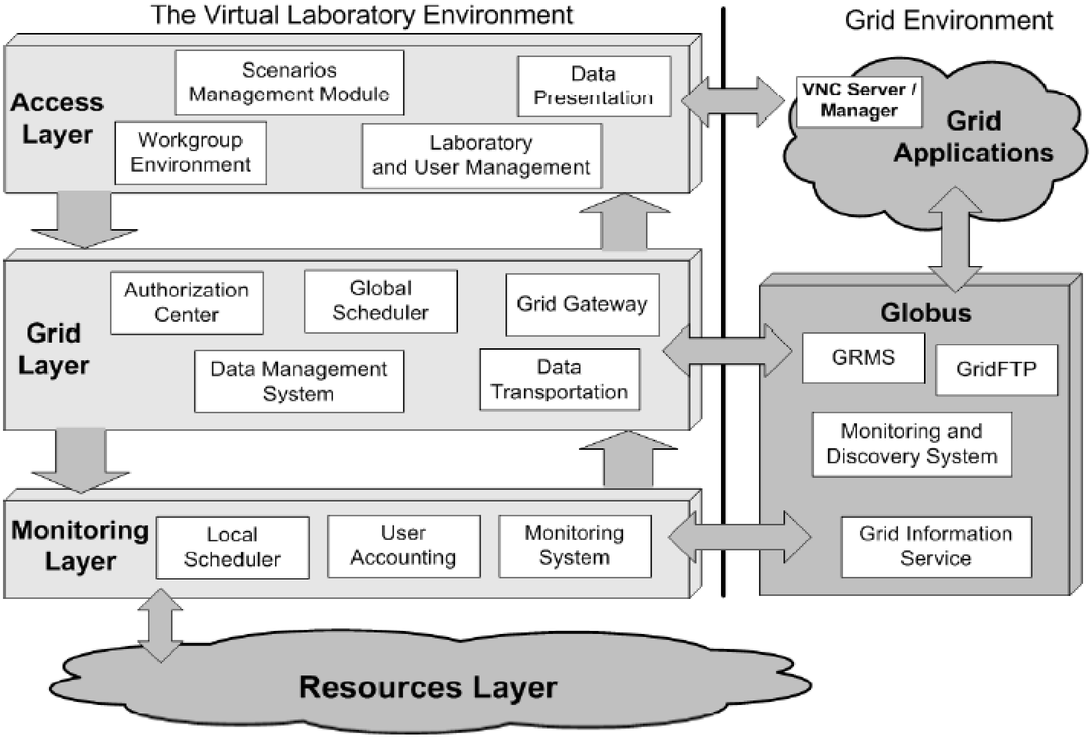


Figure 4.1.1: General architecture of the Virtual Laboratory.

An Access Layer is the top layer of this structure. Basically, it contains all modules and components responsible for a VLab user access and graphical interface to the system (including a web portal), and data input interface. Below there is a grid layer, which communicates with external grid environment. It is responsible for user authorization and authentication, data management, general task scheduling. Modules from this layer also handle the transfer of the computational tasks to the Globus system, and gather feedback data and the computed results. The Monitoring Layer consists of lower-level modules, such as hardware dedicated schedulers, system monitoring, accounting data gathering etc. All the physical scientific devices are located in the Resources Layer, as well as modules responsible for their direct control.

On the grid environment side the most important element is the Globus system, with all its components (GRMS, GridFTP etc). Globus also allows to execute computational tasks (batch and interactive) on a wide variety of grid applications.

4.1.2. Task Scheduling in the Virtual Laboratory

In the Virtual Laboratory there are two basic types of tasks: computational and experimental. Experiment is the specific task type for the VLab itself. It is scheduled to be run on scientific hardware and create a very challenging set of difficulties which need to be addressed (see p. 6.3). In this paragraph, we will focus on computational tasks, batch and interactive. Interactive tasks need a definitely more sophisticated approach. The time in which the application GUI performing the task is presented to the user (or in other words: task execution time) has to be known in advance, and synchronized with user time preferences. The mechanism responsible for displaying the interface has to be carefully designed, to address authorization and security issues. All operations performed on batch and interactive tasks are presented in the paragraphs below.

4.1.3. Batch Jobs

A general diagram describing the flow path of batch computational tasks is presented in figure 4.1.2.

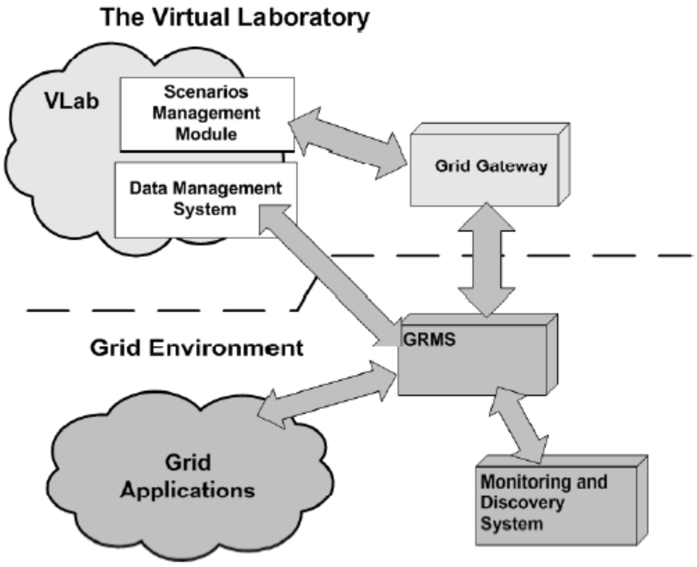
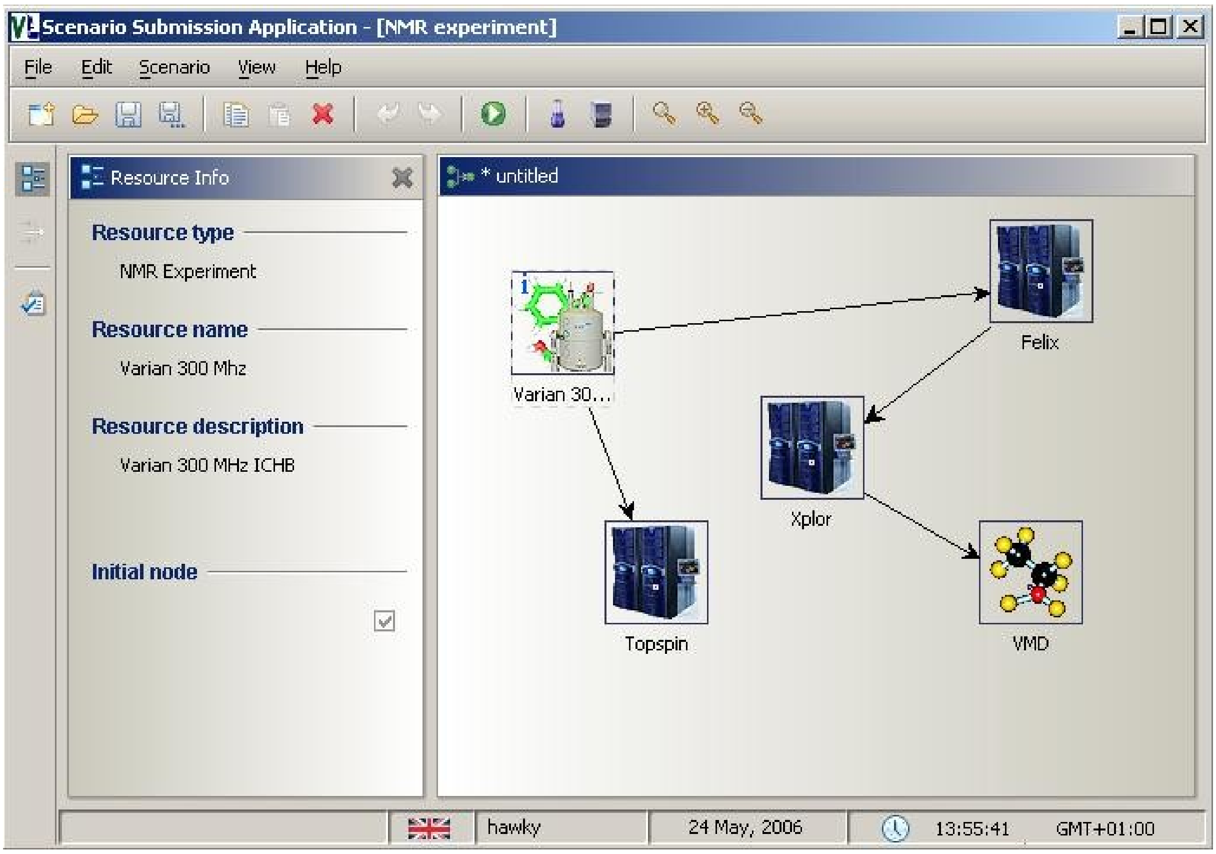


Figure 4.1.2: The batch computational tasks processing in the Virtual Laboratory system.

The computational task is created in the Scenario Submission application (figure 4.1.3) — a part of the VLab portal, responsible for creating Dynamic Measurements Scenarios. The dynamic scenario is submitted to the Scenario Management Module (SMM). At the appropriate time, during the scenario execution phase, the SMM sends the task to the Grid Gateway module via the Global Scheduler (not shown on the diagram). Grid Gateway sends the task description to the GridLab Resource Management System — an external meta scheduling system, responsible for scheduling and executing the task using the grid computational resources. The input data are passed as references to the Data Management System — the same system is used for uploading the computational results. The online task monitoring and status notification is handled by the GRMS, which contacts the Grid Gateway with the updated information about submitted tasks status.



Resource type

NMR Experiment

Resource name

Varian 300 Mhz

Resource description

Varian 300 MHz ICHB

Initial node

☑

Figure 4.1.3: Scenario Submission application.

4.1.4. Interactive Tasks

Handling the interactive tasks is more complicated than the batch ones. Different phases need to be specified here. They are described below.

Task submission. Interactive tasks are submitted into the system just as the batch ones. Each consecutive step on the diagram was marked. The detailed description is as follows:

1. A task (as a one element of the measurement scenario) is sent to the Scenario Management module from the web portal (Scenario Submission application).
2. Task is added to the database (via the Monitoring module).
3. At the appropriate time task is sent to the Global Scheduler (GS).

4. GS authorizes the user in the grid authorization module.
5. GS checks with Accounting whether the user has not exceeded the limit and can submit the task.
6. Task cannot be submitted: Inform Monitoring module. Task can be scheduled: Send it to the Grid Gateway (GG).
7. Task is transferred to the GRMS, which schedules the task and sends the information about the results (success/failure) to the Grid Gateway. The GG sends a query about the chosen server, and signs up for notifications concerning the scheduled task. If a task cannot be submitted, inform the Monitoring module.

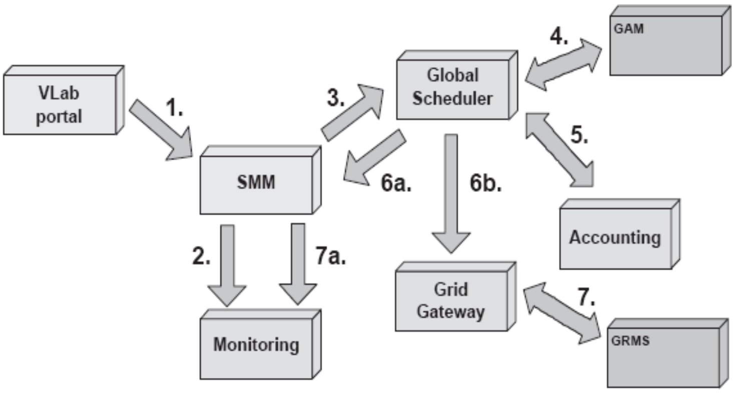


Figure 4.1.4: Interactive tasks submission.

VNC session scheduling. In the previous step the interactive task was submitted into the GRMS. Now the GRMS has to decide where the interactive task should be executed. It also reserves a slot for VNC session. Information about the session schedule is returned back to the Grid Gateway (as notifications), which in turn sends it to the Monitoring module. The detailed operations are described below:

1. The GRMS contacts the MDS system to gather information required for the connection preparation (maximum number of open VNC sessions, etc.).
2. It also contacts a VNC Session Database to check the actual sessions state. Taking all those information into account it will reserve a VNC session slot for interactive task invocation and update the database.
3. Scheduling information is passed as notification to the GG, which updates the task status, via the Monitoring module (see figure, point 4).

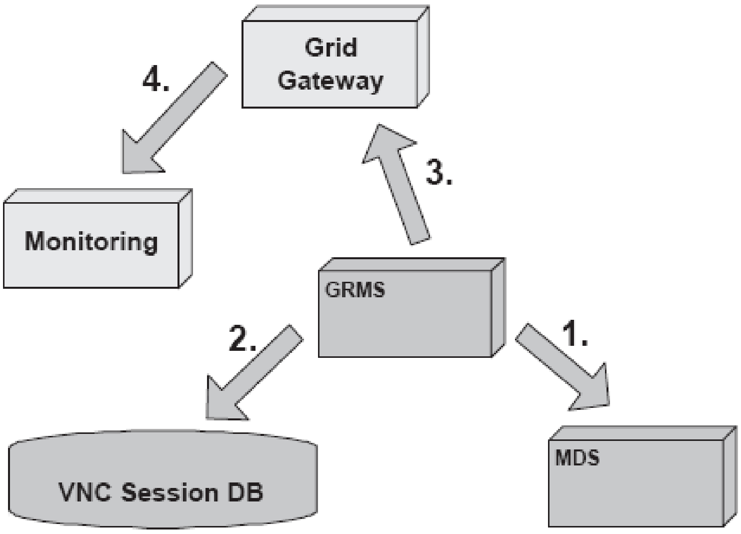


Figure 4.1.5: VNC session scheduling for interactive tasks.

Establishing a secure connection. The interactive task has been submitted to the system and the VNC session has been scheduled. The next step is to prepare the proper environment for the given task, launch it and wait forconnection establishment from the VLab user.

1. The GRMS launches its scheduled task. The task is defined as an instance of the VNC Manager which looks up the available port, runs the VNC server and the application.
2. The Grid Gateway is notified that everything has been prepared and the session can be established.
3. VNC Manager reports the port number in use and dynamically-generated password to the Grid Gateway.
4. The Grid Gateway propagates all gathered information to the Monitoring system.
5. The VLab user starts the SVNC Viewer with all the connection parameters taken automatically (and transparently) from Monitoring (and therefore has a full access to the application).

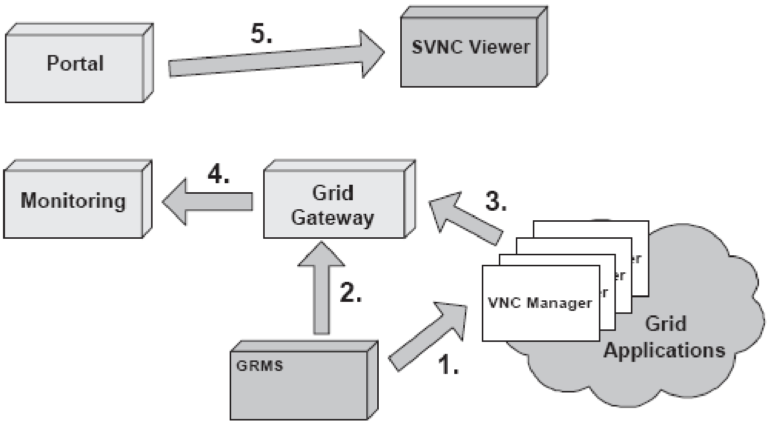


Figure 4.1.6: Establishing a secure VNC connection.

Prolonging the session. The interactive task can be scheduled for the certain amount of time. The time period is specified by the user during task definition and submission. It may happen that the reserved time slot is too short to complete the data processing. When the reservation period is about to expire, the VLab system displays the appropriate warning and the user is given the possibility to request the session prolongation. After evaluation the actual session state (VNC Session Database) the prolonged access is granted or the request refused.

Prolonging the VNC session step by step:

1. Request is forwarded to the Global Scheduler.
2. Global Scheduler authorizes the request in the GridAuthentication Module (GAM).
3. GS checks the user limits in the Accounting module.
4. Verification failed: Monitoring is informed that the session extension has failed and the process ends. Verification positive: Extension request can be sent to the Grid Gateway.
5. Request (authorized in the VLab system) is sent to the GRMS.
6. Next, the actual sessions state is looked up in the VNC Session Database and new session slot is reserved (if available).
7. Response is sent back to the Grid Gateway (as notification). GG updates the Monitoring with the new information (point 8 in the figure).

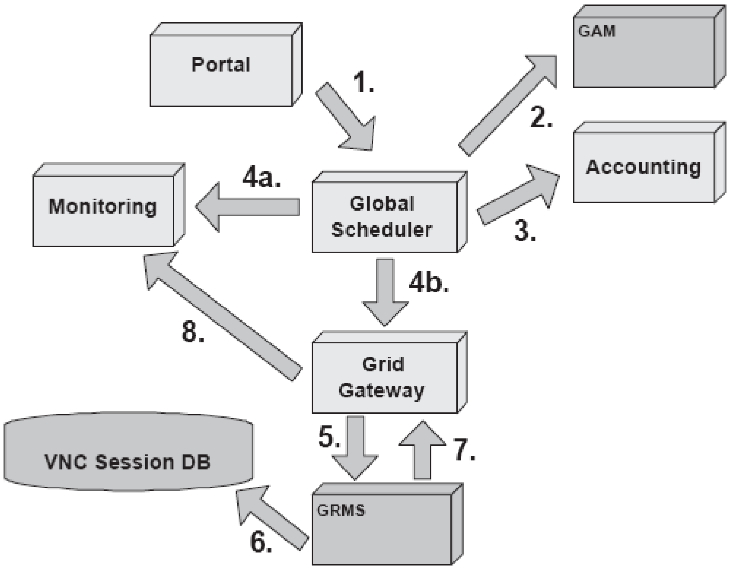


Figure 4.1.7: Prolonging the VNC session.

Finishing the VNC session by the user or by the system. The VLab user has the ability to end an active VNC session at any time after the session has been started. The procedure is explained below:

1. The request is forwarded to the Grid Gateway.
2. The GG sends it to the GRMS.
3. The GRMS system sends the appropriate signal to the instance of VNC Manager responsible for a given application. Application is closed and resources are freed.
4. The GRMS updates the VNC Session Database — registration is removed.
5. The GRMS sends the result of this operation to the Grid Gateway asan notification.
6. The Grid Gateway updates monitoring information.

Furthermore, an active VNC session can be terminated by the VLab system (and GRMS) when the reservation period expires, or for any other reason. The procedure is very similar to the described above, with the difference that points 1 and 2 are replaced by:

1. After the reservation time expires, GRMS sends a signal to end the application (via VNC Manager)
2. Then points 3–6 from the description above.

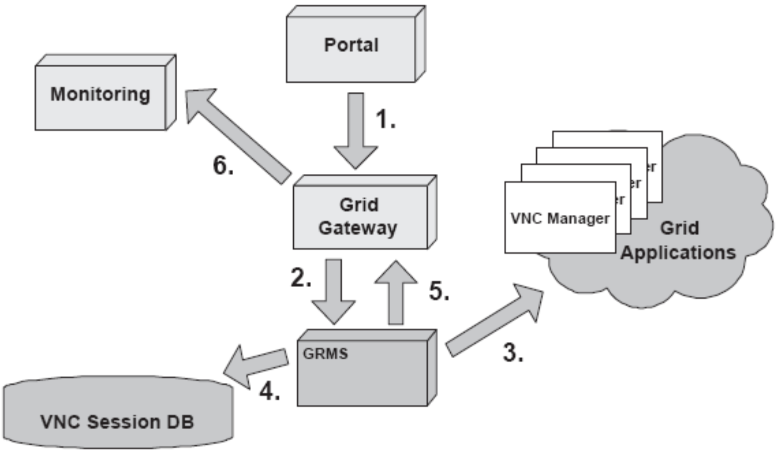


Figure 4.1.8: Ending the VNC session by the user.

4.1.5. Digital Science Library

The VLab Digital Science Library (DSL) is used to store and share experiment results, publications and manage computational data between scientists. DSL is based on the Data Management System (DMS) developed under the Progress project.

Current VLab installation implements DSL for Nuclear Magnetic Resonance Spectroscopy (storing data from NMR experiments, such as spectra, compounds, shifts etc.).

DMS is considered as an internal service supporting grid computing. The main task of this system is storing and managing the computational data. One of the additional functions is proxy to other databases and data accessed from the Internet. The next function is mirroring the whole or a part of the external database. DMS consists of three logical layers. The highest layer of the DMS is a metadata repository that keeps information about the data managed by the system. The second layer consists of the Data Broker and the Mirror & Proxy. The Data Broker is an interface between the external client and DMS. The client (e.g. external service) can be a grid broker or special software DMS client e.g. computational web portal. The Data Broker can also arrange several replicas of data. The mirror and Proxy is responsible for accessing and mirroring external databases. The main purpose of this module is to grant access to various external objects in a unified way. The lowest layer is the Data Storage. This layer is responsible for the physical storage of data. This module can store data in several ways: in computer filesystems, in databases or on tape storage systems.

The whole system uses the "web services" approach to co-operate between all modules and other components of the grid. Great emphasis was placed on the design system according to other data management systems and the proposed standards from the international bodies. The described structure is shown in figure 4.1.9.

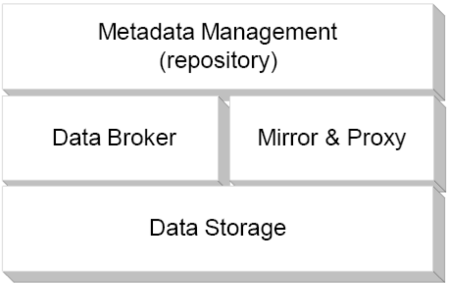


Figure 4.1.9: Logical architecture of the DMS.

Metadata Management and repository. The Metadata Management module (MD) is one of the most important modules. You can think of the repository as a place where metadata are stored. The Metadata Management module is responsible for storing and manipulating metadata. The data are described with attributes and access rules. In fact the metadata are stored in an object database in a "property = value" format. This format can be defined by user or chosen from the predefined formats like the Dublin Core (DC) standard. We decided to run the MD module as a single instance service. This decision was made because of complexity with handling metadata information which is stored in the MD module. Running multiple instances would generate issues, which concern problems with data consistency and reliability. Consistency and reliability of metadata can be provided by a replication mechanism in the underlying database system.

Data Broker and Mirror and Proxy module. The Data Broker (DBR) is an access point of the DMS system. The key functions of this module are: serving the client requests with authorization and managing of replicas in DMS. DBR module is the access point for all other services or users requesting data operations. This module can be run in multiple instances to assure reliability and good scalability. All kinds of requests addressed to the DMS system flows through the DBR module. The system can use multi instances of this module. This approach provides the system with efficient data processing. Because of a need for client authorization procedures in the DBR module it is possible to create local system access polices for users. Polices are managed by organizations which runs the DBR module on their resources. The DBR module manages data replicas that are stored in Data Storage modules. The main function of the Proxy module is to relay communication between a client and shared resources in the Internet. This module caches frequently accessed resources and offers uniform interface. This module stores the cashed data with special attributes in the Data Storage module. The basic functions of the Mirror module are to care about the consistency of data and mirror the earlier defined set of data. This module can use the remote resources through the proxy module.

Data Storage. The Data Storage module enables access to physical data. The data are arranged in data containers and can be stored on all media types and accessed by uniform interface. The data can be organized as files on generic filesystems, BLOBs in databases or files on data tapes. In the distributed scheme Data Storage modules can store data in one or more media. The most important functions of this module are: reading and writing data, providing data security, sharing data via protocols: GASS, GridFTP and FTP. The DMS system can use multiple instances of the DS module. This approach enables uninterrupted data accessibility even in case of network connection failure or system crash on which the DS module operates. This module is also capable of finding optimal DS module for client connection.

4.1.6. References

Okoń, M., Lawenda, M., Meyer, N., Stokłosa, D., Rajtar, T., Kaliszan, D., Stroiński, M.:Interactive task invocation in the Virtual Laboratory. The 4th Cracow Grid Workshop, ISBN 83-915141-4-5, pp. 293-300, Cracow, Poland, December 12-15, 2004

4.2. GridCC

4.2.1. Architecture

The GridCC EU project [GRIDCC] was launched in September 1st 2004 and lasts for 3 years, aiming at integrating scientific instrumentation and other equipment with the grid. The project's main contribution is the Instrument Element (IE), an abstraction similar to the Computing Element and the Storage Element of the classical grid. Additionally to that, the project studies methods to provide QoS guarantees to service consumers of IEs (or other resources, when that is possible) and to enable building and execuring complex scientific workflows which are QoS-enabled. It also comes with a novel Multipurpose Collaboration Environment (MCE) which includes an integrated Virtual Control Room (VCR) for the control of remote instrumentation. A host of applications are being developed based on the GridCC middleware (or ported to it), as part of the project.

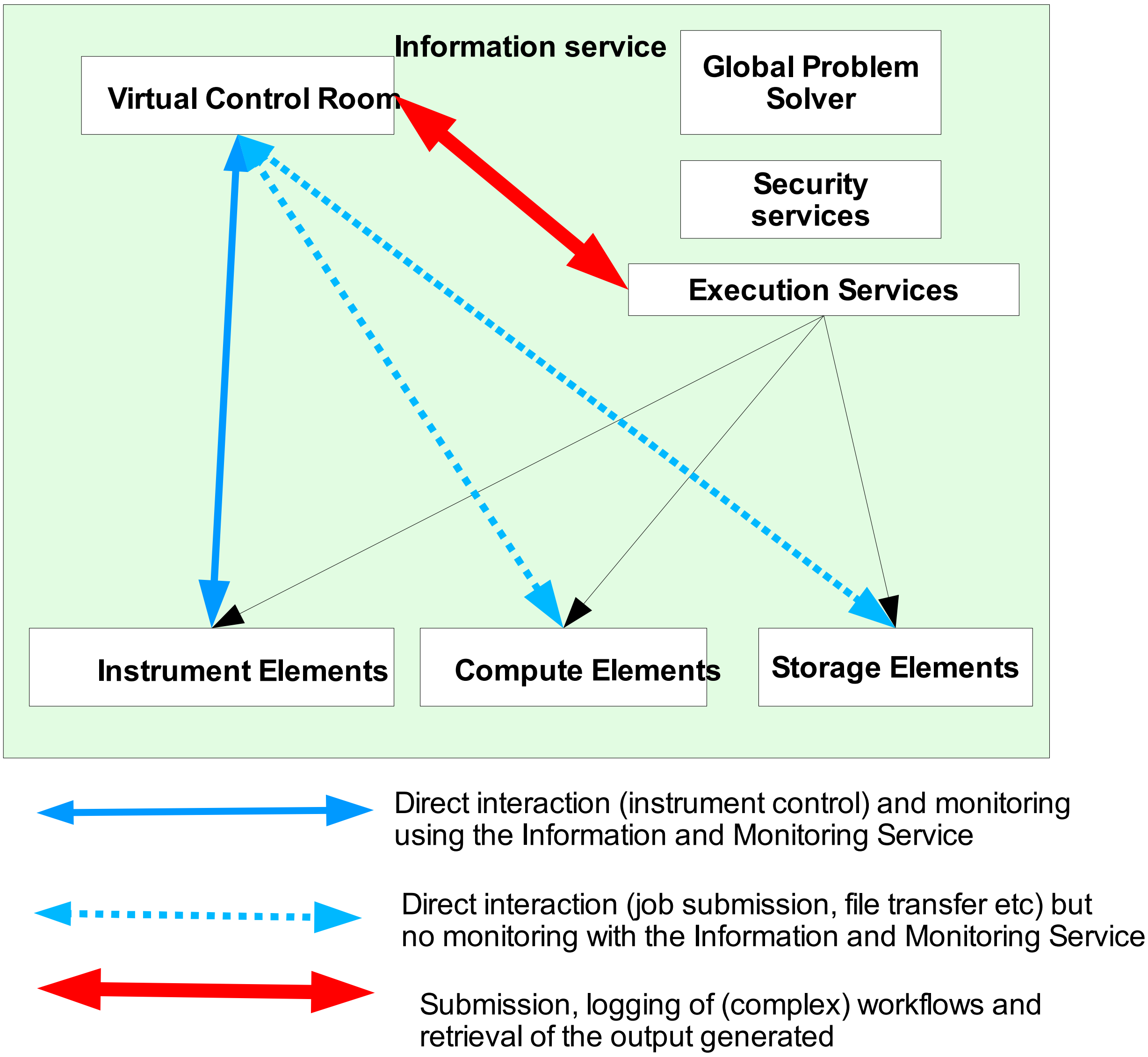

Information service
Virtual Control Room
Global Problem Solver
Security services
Execution Services
Instrument Elements
Compute Elements
Storage Elements
Direct interaction (instrument control) and monitoring using the Information and Monitoring Service
Direct interaction (job submission, file transfer etc) but no monitoring with the Information and Monitoring Service
Submission, logging of (complex) workflows and retrieval of the output generated

Figure 4.2.1: GridCC architectural overview.

A rough illustration of the GridCC m/w architecture is available in figure 4.2.1 [GRIDCC-ARCH]. The GridCC m/w contains a number of new components:

- The **Virtual Control Room** is the GridCC user interface. It allows users to build complex workflows which are then submitted to the Execution Services (ES) though the Workflow Management System (WfMS) entry point. The VCR can connect to resources directly and be used to control instruments in real time. It also extents human interaction with grid resources through its Multipurpose Collaborative Environment. The VCR is an instance of the MCE for a specific GridCC application.
- The **Instrument Element** consists of a coherent collection of services which provide on their own all the functionality to configure, partition and control the physical instrument behind the IE. In reality an instrument can actually consist of a logical collection of instruments and it is the responsibility of the instrument owner to decide how to group their instruments. Within the IE it is possible to have a Local Problem Solver (LPS) that is capable of diagnosing and reacting to error conditions in such a way as to protect the physical instrument from harm. The performance Information and Monitoring Service (IMS) is critical to the successful operation of theIE.
- The **Compute and Storage Elements** (CE and SE) within GridCC are similar to those within other grid projects. However these are extended to encompass the quality of service requirements that are central to GridCC.
- The **Execution Services** (ES) are at the heart of GridCC. The Execution Services include: the Workflow Management System for workflow execution and lifecycle management, the Workload Management System (WMS) for service discovery, job submission, logging and book-keeping, and the Agreement Service (AS) for reservation management. The ES control the execution of the workflows defined by the user in the VCR and they maintain the status of the tasks that make up the workflow. This information is passed to the user in the VCR upon request. The ES also control the quality of service, as required, by making resource reservation agreements with the resources available to it whether these beIEs, CEs or SEs.
- GridCC requires an all pervasive **Information and Monitoring System** (IMS). The performance of this service is critical to the successful operation of the instruments within the GridCC grid. The IMS is eventually expected to work using a publish and subscribe model, passing information to the different components as they require them. The IMS also passes information between the different services that make up IE. As well as IMS there is a need within GridCC for an Information Service (IS) that passes less time critical information. This is to be used by the ES (and the VCR) to discover available and appropriate resources.
- **Problem Solvers.** Automated problem solving in a grid environment is a feature of the GridCC project. Two levels of problem solver can exist within the GridCC grid: the first local to a given IE, the Local Problem Solver (LPS). The second global to the entire system, the Global Problem Solver (GPS). The LPS solve problems related to the function of a given instrument and are embedded within that IE, whereas the GPS solve problems relatedto the whole system.
- **Security System.** The GridCC applications have strict security requirements. Yet the security system is constrained both to being light-weight, so that the QoS requirements can be met, and to inter-operate with security systems of other grid systems. In order to satisfy these two requirements, GridCC has decided to use a split security system. When interacting with the components of other grid projects the GSI security will be used and the user identified by their X.509 proxy certificate, however when interacting with the IE the user will be identified by a Kerberos ticket. The level of security provided by

both systems is equivalent, however the symmetric Kerberos ticket based model can function much more rapidly. The roles that any individual can take at any given time will depend upon which Virtual Organizations (VO) they are members of.

4.2.2. Identified Use Cases

The GridCC project has four major use cases [GRIDCC-TA]:

1. Compact Muon Solenoid application
2. PowerGrid
3. meteorology
4. accelerator facility

4.2.3. Compact Muon Solenoid (CMS) Application

This application involves the use of the grid in a real-time environment to control and monitor remote large-scale detectors. This application will make use of a High-Energy Physics (HEP) experiment, the CMS detector which is currently under construction at the future LHC collider at CERN. CMS consists of 20,000,000 electronics channels that will be read out by a complex distributed data acquisition (DAQ) system feeding a large processor farm charged with filtering an input rate of up to 100 kHz down to only ~100 Hz of physics events. The DAQ system involves a very large number (a few thousand) of intelligent modules and computers, with data throughputs of ~100 Gbytes/s. These characteristics, along with the selectivity of one event in 1,000, are unprecedented in the field, and introduce requirements on the control and monitor of the experiment's data-taking.

The so-called "Run Control" and "Detector Control Systems" of the experiment are charged with supervising the full configuration of the detector, but also with monitoring the data read out, their analysis and on-line interpretation. Both systems are of paramount importance for the correct operation of the experiment. Monitoring this detector, and potentially changing settings as a result of analysis on the monitoring data, is a complex task shared by a few hundred people distributed in geographically distributed laboratories. This task requires continuous analysis and display of large amounts of data generated by the detector and in the past was done in a counting room near to the detector. In the context of GridCC, this application will be made to run in a completely distributed fashion, over the grid.

4.2.4. PowerGrid

In electrical utility networks (or power grids), the introduction of very large numbers of 'embedded' power generators often using renewable energy sources, creates a severe challenge for utility companies. Existing computer software technology for monitoring and control is not scalable and cannot provide a solution for the many thousands of generators that are anticipated. GridCC technology would allow the generators to participate in a Virtual Organization, and consequently to be monitored and scheduled in a cost-effective manner.

A specific testbed application will be built and demonstrated within the GridCC project by means of computer simulation and emulation. Existing software at Brunel University will allow the real-time simulation of a representative power network and the associated generators. New software will be created to interface the generator simulations to the GridCC environment. Distributed generator scheduling algorithms will be modified to utilize GridCC technology. The test bed will demonstrate the performance of the emulated system under various conditions, ranging from light power system loading (where energy economics is most important) to power system emergency conditions (where overloaded power circuits necessitate co-ordinated generator control to avoid power black-outs).

4.2.5. Meteorology

"Ensemble Forecasting" has been used at large meteorological centers worldwide (e.g. ECMWF, NOAA/NCEP, UK-Met Office, METEO France) with promising results. However "Ensemble Limited Area Forecasting" is still in its infancy. The main reason for this is the demanding requirements for computing resources. These resources are nowadays both available and manageable on the grid. With real-time extensions, Limited-Area Forecasting can become a common tool.

Ensemble weather forecasting on a limited area is a near real-time application because it requires the forecasting products to be ready in a few hours, at most, after the collection of the necessary observational data and initial guess fields (preparation of initial and boundary conditions). The application being developed and tested in the context of GridCC is a direct real-time port of this data, its analysis and the presentation of the results in an interactive fashion.

4.2.6. Accelerators

Far remote operation of an accelerator facility (i.e. the Elettra Control Room in Italy) involves the planning of accelerator operations, the maintenance of the accelerator and its troubleshooting, the repair of delicate equipment, understanding and pushing performance limitations, performing studies, performing commissioning and set ups and routine operations. All these activities are based on large amounts of information, which are at present accessible only at the accelerator site.

Far remote operation combines elements of immersive (i.e. providing the feeling to be present at the remote location) communication and cooperation technology. This includes video and audio presence, allowing the simultaneous operation of the same instruments, having access to the same accelerator controls and the relevant data, meeting easily and spontaneously and providing full awareness ofthe presence of the collaborators.

The security and networking issues include adequate user management, session and floor control, and secure, synchronous and reliable data transmission and distribution.

4.2.7. Details of Special Interest

The GridCC middleware, being by definition focused on connecting scientific instrumentation on the grid, is very important to RINGrid and extracting results for our studies. The most important contribution of GridCC is the Instrument Element abstraction and the work on QoS (QoS-enabled workflows, resource reservations, predictive QoS for web service invocations). The tools provided by the MCE for collaboration among scientists handling sensitive and expensive equipment are also important for unifying instrumentation with the grid.

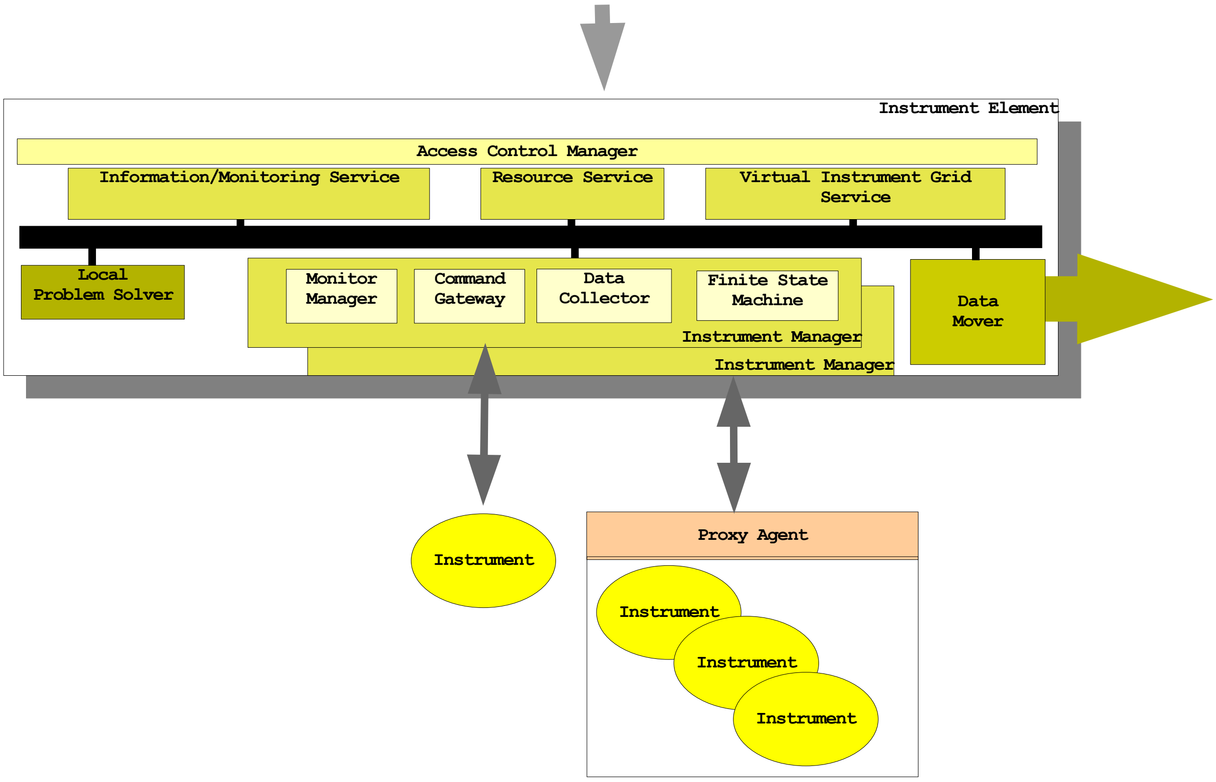


Figure 4.2.2: Instrument Element architecture.

Figure 4.2.2 displays the architectural details of the IE. The general idea is one typical in the case of device control, where properties can be set and then operations be called to use those properties. The IE offers a secure entry point (kerberos-enabled) to access the set of services. At the other end of this component, there is the Instrument Manager to interact with the real, physical instruments. The IM is one part of the software that needs to be modified depending on the application and the device to be used, as it needs to interact directly using a vendor-specific API. Each instrument manager is composed of four subcomponents:

- **Command Gateway** to interpret control requests,
- **Monitor Manager** to collect monitoring data and provide it to the IMS and the LPS,
- **Data Collector** to collect domain-specific data from the instruments and possibly pass them on to the Data Mover,
- **Finite State Machine** to reflect the collective state of the instruments controlled by the specific IM.

There also exist a set of complementary services within the IE for problem solving and predictive functionality on a local level (Local Problem Solver, LPS), for movement of instrument data to storage or computing elements (Data Mover, DM), for delegation of monitoring data to external interested parties (Information and Monitoring System, IMS), for managing the catalogue of instruments (Resource Service, RS) and for parsing control requests (in collaboration with the RS) to be delegated to the appropriate IM (VIGS).

The QoS-related part of the middleware, handled to the largest extent by the Execution Services, comprises of a a repository to hold performance data for remote service operation calls, a QoS-aware Workflow Management System (WfMS) and a service to perform resource reservations in a uniform manner. The repository (not described within the architecture document) keeps performance information for the different stages in the lifetime of a remote operation's call. Although this has been built with web service calls in mind, it can be adapted to other types of remote calls. A statistical methodology has been applied to experimental data and has successfully provided upper bounds for operation executions, thus allowing for soft QoS guarantees. Performance data is collected by clients, services and network probes. Such information is to be utilized by the WfMS, in combination with the Agreement Service (AS), for hard QoS guarantees through resource reservation. The AS provides a single web service interface for requesting reservations, although there are different domain-specific languages used to describe the reservation terms. Storage and instrument (element) reservations are already available, with computing element reservations being designed and planned for the future. The WfMS will be the main client of the AS, alongside the Workload Management System (WMS), which is the broker of submitted single (atomic) tasks. An overview of the Execution Services and how it fits with the VCR and the IS is provided in figure 4.2.3.

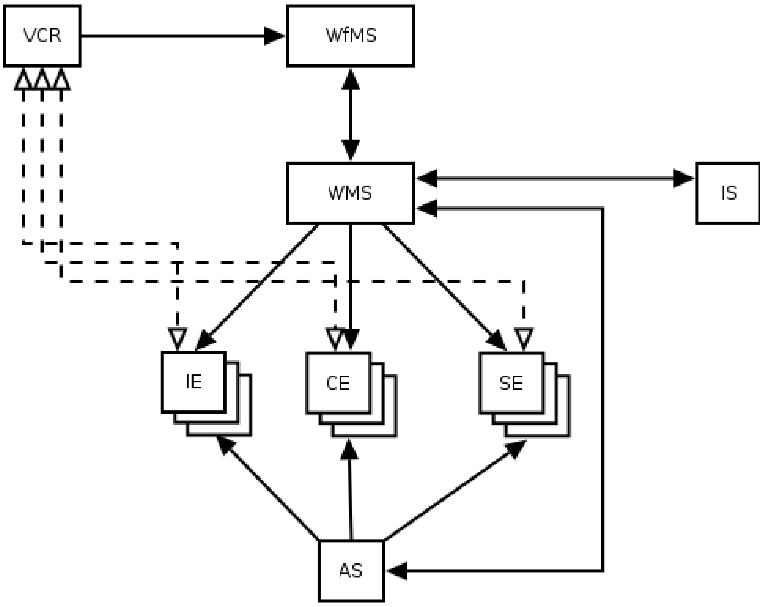


Figure 4.2.3: Execution Services architecture.

5. User Requirements

5.1. Introduction

So far this document discussed in detail middleware components and related infrastructure which should be possibly used when sharing scientific instruments via international networks. However there is little evidence how useful the middleware is. To get that information, one has either to look at the instruments and find out what technical requirements have to be met so that they can be operated successfully. This is the path WP2 has taken to some degree. However, this is a cumbersome process because there are so many instruments available and one or the other instrument will always be missed. So, we are proposing to do it the other way round: We choose several typical applications of these instruments and try to derive some constraints the middleware has to fulfill. When taking this approach, it could happen that we find out that some instruments cannot be operated successfully with the current technology. This is what WP4 is for: WP4 tries to fill the gaps we are pointing at so that WP6 can achieve the overall project's goals. Figure 5.1.1 shows how WP2's approach can be mapped to our approach.

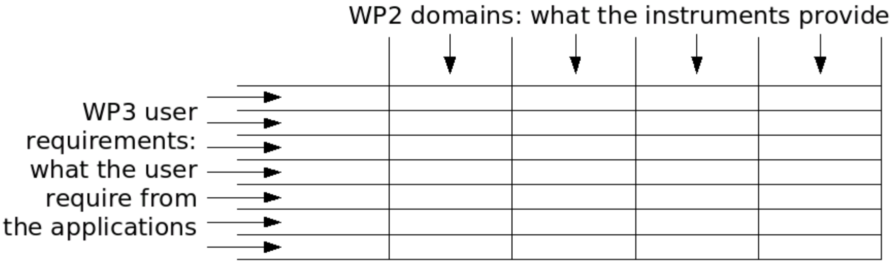

WP2 domains: what the instruments provide
WP3 user requirements: what the user require from the applications

Figure 5.1.1: WP2 domains and WP3 user requirements.

5.1.1. Criteria for Successful Middleware Selection

Remote instrumentation has very distinct requirements, as laid out in section 1.2. In particular, the next four sections give an detailed explanation what the user expects in each of the scenarios with respect to:

- **data rate** (several thousand per second, one sample every hour, cf. sensors)
- **timeliness** (real-time processing required because of emergency responses or analysis at a later time possible?)
- **value** (of each individual measurement, is loss of a sample critical?, depends more or less on the number of data sources)
- **storage space** (how much space is needed, have distributed filesystems to be utilized?)
- **interactivity** for visualization of experimental data or the control interface (latency and jitter are probably critical)
- **collaboration tools** (are they scalable for a large user base?)
- **high-volume data transmission** (also see deliverable 3.1 for this)

5.1.2. Selection of Representative Applications

In order to get representative conclusions, the applications have to be chosen carefully. It is best when applications fulfill all of the criteria mentioned in the previous section to a varying degree. If this is the case, most of the possible applications are present for determing the suitability of a certain middleware component. We have identified the following application areas:

- measurement, control and automation (MC&A)
- large-scale physics and astronomy
- sensor networks
- nuclear magnetic resonance (NMR) spectroscopy

As it is shown by different color codes in table 5.1.2, the applications are highly different as far as their requirements are concerned.

	measurement, control and automation	**large-scale physics and astronomy**	**sensor networks**	**NMR spectroscopy**
processing power	some ("smart")	high	moderate (RISC microprocessors)	high
operated manually/ automatically	manually	manually	automatically (ad-hoc multihop network)	manually
cost per element	small-medium	expensive	low-moderate	expensive
number of data sources	relatively high	singular	large	singular
operation mode	near real-time to real-time	real-time	normal (not critical)	normal (not critical)
data rate	low-moderate	high	low-moderate	moderate
data value	relatively low	very high	low	high

Table 5.1.2: Chosen applications and selection criteria.

In the following four sections, the requirements for each of the identified applications are evaluated.

5.2. Requirements for Measurement, Control and Automation Equipment

The category of instrumentation classified in commercial terms as MC&A refers to test and measurement equipment that can be broadly defined as the bulk of instruments and sensors used to test, measure, analyze, control, calibrate, display and record data in laboratory and other testing situations (http://testequipment.globalspec.com/ProductFinder/Labware_Test_Measurement). This includes measuring instruments (temperature, pressure, flow, level and other parameters); chemical analysis instruments; recording and display instruments; controllers and control systems; supervisory and communications systems; process control software products; testing and maintenance instruments.

Among the characteristics of this type of instrumentation that can be useful to define requirements in broad terms, as regards networking and middleware functionalities for their remote control, a few are particularly noteworthy:

- In the majority of cases, the devices that are involved belong to a small-medium scale (as compared, for instance, with large-scale scientific equipment, such as particle accelerators and radio-telescopes); however, they exhibit processing capabilities to a certain extent or, in other terms, they are "smart" instruments, as opposed, at another extreme, to simple sensors;
- In the majority of cases, they are manually operated or supervised;
- Their cost is at a medium to small level (again, as opposed to large-scale scientific instrumentation), but their number is relatively high;
- Their operation often requires real-time or quasi-real-time interaction.

When developing middleware for any category of equipment, it is important to identify some preliminary design characteristics. In fact, instruments form a diverse category of machines, with widely varying aspects. Thus, it is not possible to completely decouple the design of the middleware from the development of the overall structure itself. Specifically, when talking about MC&A systems, the following characteristics (intrinsic as well as extrinsic ones) [1] should be looked at:

Number of sources. The number of sources impacts the design of the information system supporting the instrument and also reflects in manageability. However, a huge number of sources is not a major issue with most MC&A, except when in the presence of sensor networks as pervasive acquisition devices.

Data rate is not a critical issue in MC&A.

Real time. Some instruments output data, or some automated process can rely on data to be analyzed later. Others require real-time processing of data. This might be because researchers are using the instrument interactively, or because a control loop must be closed to control a plant. This is the classical case of tele-controlling remote plants by using remote controller. Most MC&A instrumentation must be carefully considered from this point of view.

Degree of accuracy. in some experiments or plants, the data of each individual source may be extremely valuable. In others, like in massive sensor networks, the loss of some readings is not catastrophic. Middleware for handling valuable data may need to incorporate fault-tolerance or redundancy into its design.

Besides intrinsic characteristics, other factors can affect the design of information systems for instruments. These factors are determined by the hardware design, and are termed extrinsic characteristics. Extrinsic characteristics may change as the hardware changes and the middleware must take care of it, by providing information hiding features. This is mandatory in order to minimize the need of rewriting important portions of code when the device technology varies. In other words, the effort needed to tweak the middleware as the underlying device technology (and not the task itself) varies must be minimized. Specifically:

Processing power. Systems based on MC&E instrumentation must perform a variety of computational tasks, ranging from participating in network protocols to numerical analysis. These tasks may be divided among a variety of nodes. Some nodes may consist of little more than an analog-to-digital converter. Others may be powerful special-purpose computers. Taking inspiration from overlays available in P2P networks, the next generation of middleware must provide a proper degree of device independence, e.g., by providing a proper service of load distribution and data collection.

Heterogeneity of resources. Each device may have a varying amount of storage, while some instruments may have very little bandwidth available. In situations where there is little bandwidth headroom, a proper portion of the controlling middleware must accommodate the required data rate, perhaps by techniques such as compression, which will in turn require more processing power.

With such requirements, next generation middleware will be a complex piece of software. In addition, in order to have a feasible development cycle and a maintainable code base, future middleware must be modularized. Then, different components will perform different tasks. Nevertheless, to foster the interaction among different communities (e.g., scientist, physicist, software engineers, …) modules must be based on solid standards and communicate through well-known protocols (e.g., the Session Initiation Protocol, SIP).

In more details, future middleware must be:

Independently deployable. A middleware component can depend on another component for correct operation, but it still must be possible to install and upgrade it individually. Furthermore, this deployment should not require programming skills.

Reusable by third-parties. Components can be reused in different applications by the end user. Such interchangeability is crucial to making them effective at coping with rapid change.

Connectable by third-parties. Components can be connected to one another in a fashion similar to stereo components, or hardware modules. This allows the end user to solve completely new problems from existing components.

Large granularity. A component generally encapsulates more functionality than an object. Most components will be implemented internally with many objects.

Inherently distributed. Remote operation is an inherent characteristic of components. This means that local operation is a special case of normal operation, rather than vice versa. This characteristic forces a degree of encapsulation that often makes components easier to use even in the local case.

Taking also inspiration from ongoing projects (previously cited in this document), we can summarize the requirements for future middleware for MC&A equipment as follows:

1. Support processing of data in real-time across a variety of timescales.
2. Support different rates of data production under varying available bandwidth. This will probably require the ability to integrate multiple protocols within one system or a variety of coders/encoders.
3. Provide end-to-end reliability and fault tolerance.
4. Support IPv6 natively.
5. Support large numbers of sensors or at least being able to interact with sink nodes in order to switch from fine-grained to coarse-grained measurements and vice-versa.
6. Support integrated representations of instruments, comprising disparate sensor types.
7. Support real-time fusion of heterogeneous information from many sources to create one data stream.
8. Provide functional transparency; each function of the instrument must be completely and accurately accessible.

5.2.1. References

R. Bramley, K. Chiu, J. C. Huffman, K. Huffman, D. F. McMullen, "Instruments andsensors as network services: making instruments first classmembers of the Grid," Indiana University CS DepartmentTechnical Report 588, 2003.

5.3. Large-Scale Physics and Astronomy

We are exploring issues related to remote control of SOAR science programs to maximize productivity:

1. Optimal subdivision of bandwidth between video and data by dis/re-connecting video conferencing under software control. Reconnection takes less than 5 seconds.
2. Minimizing real-time data rates from large detector mosaics by first sending a compressed rendition with sufficient fidelity for the remote observer to define regions of interest (ROI) that are then sent at full bit resolution before the rest of the image is refined.
3. Integration of telescope and site status telemetry into tactical displays for the remote observer and the telescope operator. Several of these are based on the PROAS package, written by N. Cardiel at IAS (Canary Islands).
4. Remote, low latency use of instrument control GUI's.

5.3.1. Masks and Data Rates

A powerful functionality of JPEG2000 is to send first ROI's with lossless compression, and then progressive compressions of the rest of the image. While the astronomer is interacting with data in the small ROI's (e.g. measuring the sky brightness or the object PSF, or taking a power spectrum to assess background noise), the lossy compressed full image is arriving at a rate that does not impede the fidelity of the video conferencing link. Useful masks to define ROI's are those set by levels of image brightness (e.g. "all pixels brighter than X-times the mean local sky"), connection to previous ROI's, and wavelength/spatial ranges of interest in multislit or integral field spectra. Note that the entire image may eventually arrive, but SOAR always archives the lossless compressed version in Chile for later access.

Data rates from 1st-generation SOAR instruments will average about 4.6 GB/night (1.6 TB/year), peaking at about 8.5 GB/night. Assuming 2:1 (50%) lossless compression, a night's data can be retrieved to La Serena on the 75 Mbps link in 10-20 minutes using only 1% of link capacity to preserve the rest for others. In La Serena, data will be retained in a "disk farm" running as a software level-1 (disk mirroring) RAID array. Firewire-interface disks are so cheap/GB that the entire SOAR archive can grow online by adding less than 10 disks over a year. Sending a night's data to the U.S. will require 3.3-6.9 hours on the AURA dedicated 4 Mbps Abilene link (taking 16-30% of the link capacity, so several video conferencing/data transfer sessions by other observatories can also be ongoing). The disk archive will be replicated somewhere in the U.S. and in Brazil, and its contents eventually ingested into the National Virtual Observatory.

With the telecom bubble's glut of optical fiber, we can transmit data more efficiently than astronomers and "observe remotely". With data compression, less than half of the 6 Mbps bandwidth shared currently by SOAR and CTIO is enough to enable a high-fidelity observing presence for SOAR partners in North America, Brazil, and Chile. We discuss access from home by cable modem/DSL link.

Currently, high-end VC hardware tops out at ~3.5 Mbps. The standard H.263 video compressor/decompressor (codec) is supported in hardware sold by Polycom, Tandberg, and Vcon among others for upward of $1000. We have also obtained excellent results from software codecs: a PC with at least a 1 GHz Pentium III CPU can run Vcon's vPoint. Windows software to connect to another H.263-equiped system at whatever speed is required. This software controls remote camera pan and zoom if available, gives excellent monaural sound, and allows multiple parties to connect to your PC (which acts as a Multi-Conference Unit, MCU). It works with any USB-connected camera. We use the Logitech Laptop Pro camera — it has good resolution and color fidelity. It cannot zoom or auto focus, but can be panned over a 30° arc by the remote site or locally by its excellent and accurate face-tracking software.

We have maintained a 768 kbps wireless VC to SOAR on a notebook computer carried around the UNC Physics &Astronomy Dept.

The quickest way for a distant user to control an instrument or telescope system has been to export the entire instrument console to their desktop. This has been done for more than 15 years with the X11-window system, and now even Windows XP Pro supports remote logins. However, these connections require more than 5 Mbps burst bandwidth; otherwise mouse cursor and window redraw latencies become too irritating or unreliable. Latencies are shorter at similar bandwidth in newer packages such as VNC.

Remote panels also add another layer of security and confidentiality by hiding the other resources and directories of the executing computer. To prepare for their observing block, several users can connect to a single panel without degrading performance. The Goodman spectrograph control system is an example of a LabVIEW application that is fully compliant with remote panels.

5.3.2. Data Transfer

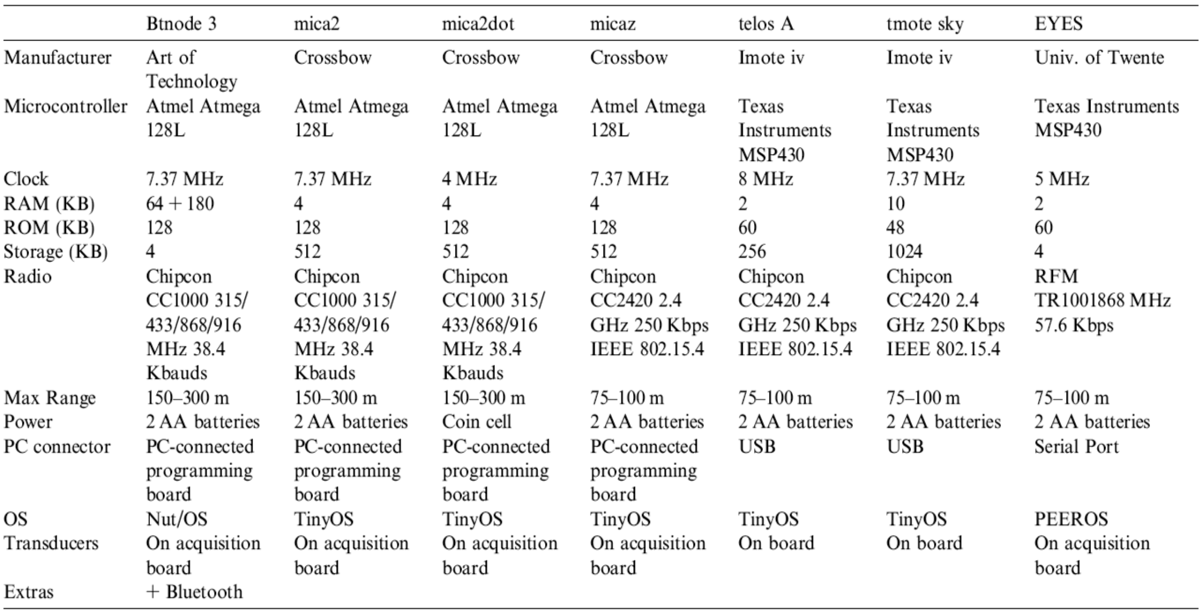
Our LabVIEW-coded Remote Display Tool (RDT) now incorporates capabilities provided by the latest version of IMAQ Vision. We now use the commercial Kakadu JPEG 2000 codec (http://www.kakadusoftware.com) instead of the freeware Jasper package (http://www.ece.uvic.ca/~mdadams/jasper). Kakadu supports SIMD optimizations, doubling speed over Jasper. With a P4 2.2 GHz PC, we compress 500-fold a typical SOAR 4K2 16-bpp image in 4.5 sec. Kakadu uses modem coding standards (C++ classes rather than C typedefs), and is supported by its author as a commercial product for $100 for academic users. RDT is coded in LabVIEW with the IMAQ library, but its executable requires only a local runtime license ($100 for academic license, Windows only). Once the IMAQ runtime license is active, the RDT can be downloaded from SOAR, and auto-installs on the user's machine. Linux or Solaris users cannot use neither RDT nor Starry Night. We program with Microsoft's .NET compilers and those in the Minimalist GNU tree (www.mingw.com). We store data on the dual-processor PC with RAID array that runs the projected display under Windows XP. This machine is linked by AFS to archive facilities and to the observer's notebook computer (which typically runs IRAF under Linux).

After experiments, we have tuned the bandwidth of each thread to maximize total upload speed. Little may change in our view of the SOAR control room, so much less of the low latency video bandwidth allocation is often used (e. g. figure 6.1.1). This gives the remote user attached by CM/DSL opportunities for simultaneous data retrieval, receipt of telescope telemetry, and instrument control. In typical use, a 768 kbps maximum VC with two upload and instrument control threads active saturates a 3 Mbps link (half these numbers for CM/DSL access). With lossless compression, a 4K2 image is retrieved from SOAR in 100 seconds (twice this for CM/DSL). By then, our next exposure is usually well underway. The combination of multiple threads and queued retrieval of compressed data is so effective that we do not bother to interrupt VC during data retrieval.

5.4. Sensor Networks

Recent advances in Microelectromechanical Systems, tiny microprocessors and low power radio technologies have created low-cost, low-power, multi-functional miniature sensor devices, which can observe and react to changes in physical phenomena of their surrounding environments. When networked together over a wireless medium, these devices can provide an overall result of their sensing functionality. Wireless sensors are equipped with a radio transceiver and a set of transducers through which they acquire information about the surrounding environment. When deployed in large quantities in a sensor field, these sensors can automatically organize themselves to form an ad hoc multihop network to communicate with each other and with one or more sink nodes. A remote user can inject commands into the sensor network via the sink to assign data collection, processing and transfer tasks to the sensors, and it can later receive the data sensed by the network through the sink. Use of this technology appears to be limited only by our imagination and ingenuity. A diverse set of applications for sensor networks encompassing different fields have already emerged including medicine, agriculture, environment, military, inventory monitoring, intrusion detection, motion tracking, machine malfunction, toys and many others.

A wireless sensor is characterised by its small size, its ability to sense environmental phenomena through a set of transducers and a radio transceiver with autonomous power supply. Current low-end sensors employ low cost Reduced Instruction Set Computer (RISC) microcontrollers with a small program and data memory size (about 100 kb). An external flash memory with large access times may be added to provide secondary storage and to alleviate the application size constraints imposed by the on-chip memory. Common on-board I/O buses and devices include serial lines such as the Universal Asynchronous Receiver- Transmitter (UART), analog to digital converters and timers. Two approaches have been adopted for the design of transducer equipment. The most general and expandable approach, as pioneered by Crossbow [5.4_28], consists in developing transducer boards that can be attached (and possibly stacked one on top of the other) to the main microcontroller board through an expansion bus. A typical transducer board from Crossbow provides light, temperature, microphone, sounder, tone detector, 2 axis accelerometer and 2 axis magnetometer devices. Alternatives include low cost versions that provide a reduced set of transducers or more expensive versions that boast GPS, for instance. Special boards are also available that carry no transducers but provide I/O connectors that custom developers can use to connect their own devices to the Crossbow sensors. The other approach (followed by Moteiv [5.4_86]) is to put transducers directly on the microcontroller board. Transducers are soldered or can be mounted if needed but the available options are very limited and generality and expandability is affected. On the other hand, these on-board transducers can reduce production costs and are more robust than transducer boards which may detach from the microcontroller board in harsh environments. By means of the transceiver circuitry a sensor unit communicates with nearby units. Although early projects considered using optical transmissions [5.4_117, 5.4_54], current sensor hardware relies on RF communication. Optical communication is cheaper, easier to construct and consumes less power than RF but requires visibility and directionality, which are extremely hard to provide in a sensor network. RF communication suffers from a high path loss and requires complex hardware but is a more flexible and understood technology. Currently available sensors employ one of two types of radios. The simplest (and cheaper) alternative offers a basic Carrier Sense Multiple Access (CSMA) Medium Access Control (MAC) protocol, operates in a license free band (315/433/868/916 MHz) and has a bandwidth in the range 20–50 kbps. Such radios usually offer a simple byte oriented interface that permits software implementations of arbitrary (energy efficient) MAC protocols. Newer models support an 802.15.4 radio operating in the 2.4 GHz band and offering a 250 kbps bandwidth.

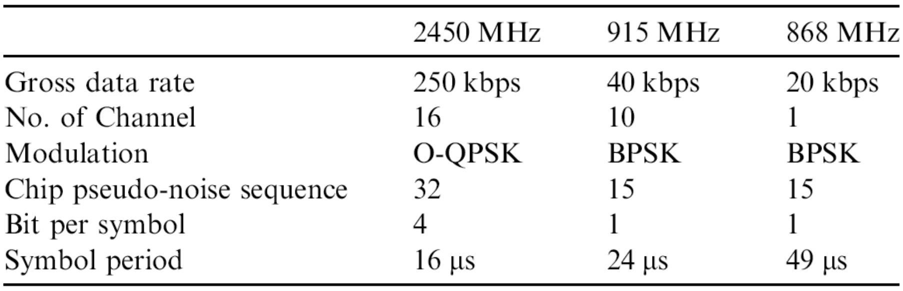


	Btnode 3	mica2	mica2dot	micaz	telos A	tmote sky	EYES
Manufacturer	Art of Technology	Crossbow	Crossbow	Crossbow	Imote iv	Imote iv	Univ. of Twente
Microcontroller	Atmel Atmega 128L	Atmel Atmega 128L	Atmel Atmega 128L	Atmel Atmega 128L	Texas Instruments MSP430	Texas Instruments MSP430	Texas Instruments MSP430
Clock	7.37 MHz	7.37 MHz	4 MHz	7.37 MHz	8 MHz	7.37 MHz	5 MHz
RAM (KB)	64 + 180	4	4	4	2	10	2
ROM (KB)	128	128	128	128	60	48	60
Storage (KB)	4	512	512	512	256	1024	4
Radio	Chipcon CC1000 315/433/868/916 MHz 38.4 Kbauds	Chipcon CC1000 315/433/868/916 MHz 38.4 Kbauds	Chipcon CC1000 315/433/868/916 MHz 38.4 Kbauds	Chipcon CC2420 2.4 GHz 250 Kbps IEEE 802.15.4	Chipcon CC2420 2.4 GHz 250 Kbps IEEE 802.15.4	Chipcon CC2420 2.4 GHz 250 Kbps IEEE 802.15.4	RFM TR1001868 MHz 57.6 Kbps
Max Range	150–300 m	150–300 m	150–300 m	75–100 m	75–100 m	75–100 m	75–100 m
Power	2 AA batteries	2 AA batteries	Coin cell	2 AA batteries	2 AA batteries	2 AA batteries	2 AA batteries
PC connector	PC-connected programming board	PC-connected programming board	PC-connected programming board	PC-connected programming board	USB	USB	Serial Port
OS	Nut/OS	TinyOS	TinyOS	TinyOS	TinyOS	TinyOS	PEEROS
Transducers	On acquisition board	On acquisition board	On acquisition board	On acquisition board	On board	On board	On acquisition board
Extras	+ Bluetooth						

Table 5.4.1: Comparison of various sensor architectures.

The latter offers the possibility of using an internal (i.e., on-board) antenna which makes sensors more manageable and self-contained with respect to an external whip antenna. The radio range varies with a maximum of about 300 m (outdoor) for the first radio type and 125 m for the 802.15.4 radios. Sensors are powered by batteries, usually a couple of standard AA standard batteries that can be replaced upon expiration. (This is important since the day of cheap, disposable sensors is yet to come.) Battery size usually determines the size of the sensor, so existing hardware is roughly a few cubic centimetres in size.

An exception is represented by the Crossbow mica2dot mote [5.4_28] which uses a coin cell about the size of a quarter dollar but is also more resource constrained than larger sensors. Studies are currently under way to replace/integrate battery sources with some power scavenging methods such as solar cells but there are some doubts about the actual effectiveness of such methods. Solar cells, for instance, do not produce much energy indoor or when covered by tree foliage. A final matter is the operating system i.e., the basic system software that application programmers can use to interact with the sensor hardware. TinyOs [5.4_122, 5.4_44] is a widely used simple lightweight event-based operating system written in nesC [5.4_39] (it is used on Crossbow motes, Moteiv motes and similar devices).



	2450 MHz	915 MHz	868 MHz
Gross data rate	250 kbps	40 kbps	20 kbps
No. of Channel	16	10	1
Modulation	O-QPSK	BPSK	BPSK
Chip pseudo-noise sequence	32	15	15
Bit per symbol	4	1	1
Symbol period	16 µs	24 µs	49 µs

Table 5.4.2: Radio front-end and physical layer specification.

It supports the task concept: An execution entity that runs to completion without being preempted by other tasks and can post other tasks. Only interrupt service routines can interrupt a running task. Lengthy operations like reading from a transducer or sending a radio message are split-phase: The requesting task invokes a command that starts the operation and immediately returns. When the operation completes code from interrupt or TinyOs routines posts a notification task. Such task calls (signals) an event routine that collects results and does other chores in user space. The command/event nature of TinyOs renders application programming rather complex and error prone. An interesting alternative comes from the Nut/OS operating system [5.4_95] that runs on Btnodes [5.4_20]. It offers non preemptive multithreading where a scheduled thread maintains processor control until it voluntarily relinquishes it, terminates or blocks on a lengthy I/O operation. Table 5.4.1 compares some existing sensor node architectures. For IEEE 802.15.4 based sensors data rate and other physical layer specification are given in table 5.4.2.

Concerning middleware architectures tweaked for sensor networks, [L1] presents many innovative guidelines and solutions for middleware development, that are partially aimed at filling some gaps as regards the communication paradigm and signaling strategies of middleware-based architectures. Besides, another interesting issue of wireless architectures concerns the capability of the nodes of self-organizing and self-configuring. The possible scenarios where sensor networks may be deployed cover a wide range of applications: from simple monitoring purposes to possible "backup backbones" for Mobile Ad-hoc NETworks (MANETs) [L2].

Furthermore, with the introduction of the support of mobility, sensor networks could be deployed for tracking moving entities, such as robots exploring a hostile environment. Wireless-based sensor networks could be exploited to monitor a zone for a short/medium time, in order to gather environmental data to be conveyed to Geographical Information Systems (GIS) or Decision Support Systems (DSS), which represent useful tools to prevent and manage hydro-geological hazards.

Clearly, such kind of networks can greatly benefit from the fact that they do not rely on a fixed infrastructure. Consequently, one of the key problems is related to the topology-building procedure that represents the first step to develop any routing algorithm.

Regarding topology-related problems, interesting studies were carried out and many others are still ongoing. Algorithms devoted to build clustered topologies are presented in [L3], while [L4, L5] offers a survey of the most popular routing strategies. In addition, even if devoted to mobile networks, [L6] offers an exhaustive survey about cluster-based algorithms.

Nevertheless, the increasing availability of cost effective and low consuming storage technologies, increased the dual role of sensors both as a "collector" and part of a distributed storage framework. In this perspective, both the Cougar [L7] and SINA [L8] systems provide a distributed database interface to the information from a sensor network with database-style queries. Power is managed in Cougar by distributing the query among the sensor nodes to minimize the energy consumed to collect the data and calculate the query result. To support the database queries, SINA incorporates low-level mechanisms for hierarchical clustering of sensors for efficient data aggregation as well as protocols that limit the re-transmission of similar information from geographically proximate sensor nodes.

AutoSec [L8], Automatic Service Composition, manages resources in a sensor network by providing access control for applications so that quality of service requests are maintained. This approach is similar to middleware for standard networks because resource constraints are met on a per-sensor basis, but the techniques for collecting the current resource utilization are tailored to the sensor network.

Furthermore, DSWare [L9] provides a similar kind of data service abstraction as AutoSec, but instead of the service being provided by a single sensor, it can be provided by a group of geographically close entities. Consequently, DSWare can transparently manage sensor failures as long as enough sensors remain in an area to provide a valid measurement. While these middleware for sensor networks focus on the form of the data presented to the user applications, Impala [L10], designed for use in the ZebraNet project, considers the application itself, exploiting mobile code techniques to change the functionality of the middleware executing at a remote sensor. The key to energy efficiency for Impala is for the sensor node applications to be as modular as possible, enabling small updates that require little transmission energy.

Although each of these middleware components is designed for efficient use of the wireless sensor network, they largely ignore the properties of the network itself. In other words, most of these approaches do not attempt to change the properties of the network in order to manage energy, and they are not flexible enough to support different protocol stacks or different applications' QoS requirements. To overcome this, an interesting solution has been developed within the framework called MILAN (Middleware Linking Applications and Networks) [L11]: When applications have the ability to adapt to changing sets of available components, MiLAN can identify these feasible sets and determine which set optimizes the tradeoff between application performance and network cost (e.g., energy dissipation). MiLAN must then configure the network so that components in the selected feasible set are linked to the application. So, a key feature of MiLAN is the separation of the policy for managing the network, which is defined by the application, from the mechanisms for implementing the policy, which is effected within MiLAN.

5.4.1. References

[5.4_20] BTnodes — A Distributed Environment for Prototyping Ad Hoc Networks. http://www.btnode.ethz.ch/

[5.4_28] Crossbow Technology Inc. http://www.xbow.com

[5.4_39] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesC language: a holistic approach to networked embedded systems, in: Proceedings of the ACM SIGPLAN conference on programming language design and implementation (PLDI 2003), San Diego, CA, USA, June 2003, pp. 1–11.

[5.4_44] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, K.S.J. Pister, System architecture directions for networked sensors, in: Proceedings of the 9th International Conference on Architectural Support for Programming Languagesand Operating Systems (ASPLOS-IX), Cambridge, MA, USA, November 2000, pp. 93–104.

[5.4_54] J.M. Kahn, R.H. Katz, K.S.J. Pister, Next century challenges: mobile networking for "Smart Dust", in: Proceedings of the 5th International Conference on Mobile Computing and Networking (MobiCom 1999), Seattle, WA, USA, August 1999, pp. 271–278.

[5.4_86] Moteiv Corporation, http://www.moteiv.com

[5.4_95] Nut/OS — The BTnode operating system core, http:// www.ethernut.de

[5.4_117] SmartDust: Autonomous sensing and communication in a cubic millimeter. http://robotics.eecs.berkeley.edu/pister/SmartDust/

[5.4_122] TinyOs Community Forum, http://www.tinyos.net

[L1] IEEE Network, Special Issue on "Middleware Technologies for Future Communication Networks", vol.18, Jan.–Feb. 2004.

[L2] S. Corson, J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", IETF, RFC 2501, January 1999.

[L3] W. Lou, J. Wu, "A Cluster-Based Backbone Infrastructure for Broadcasting in MANETs", IPDPS 2003, Proc. 17th Internat. Symp. on Parallel and Distributed Processing (IPDPS 2003), pp 221- 229.

[L4] E. H. Callaway, "Wireless Sensor Networks — Architectures and Protocols", Auerbach Publications, ISBN 0-8493-1823-8, 2004.

[L5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A Survey on Sensor Networks", IEEE Communications Magazine, pp. 102–106, August 2002.

[L6] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. IEEE Personal Communication, 7:10–15, October 2000.

[L7] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen. Querying and Tasking in Sensor Networks. In SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V), Orlando, Florida, April 24–28 2000.

[L8] Q. Han and N. Venkatasubramanian. Autosec: An integrated middleware framework for dynamic service brokering. IEEE Distributed Systems Online, 2(7), 2001.

[L9] S. Li, S. Son, and J. Stankovic. Event detecion services using data service middleware in distributed sensor networks. In Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks, April 2003.

[L10] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In ACM SIGPLAN Symposium on Principle and Practice of Parallel Programming (PPoPP'03), June 2003.

[L11] Project MILAN: http://www.futurehealth.rochester.edu/milan/

5.5. Nuclear Magnetic Resonance Spectroscopy

Nuclear Magnetic Resonance (NMR) spectroscopy is a unique experimental technique that is widely used in physics, organic and inorganic chemistry, biochemistry as well as in medicine. It is a powerful technique for obtaining structural and dynamic information on molecules at the atomic level. NMR phenomenon itself serves observing physical, chemical, and electronic properties of molecules, allows to determine spatial structure of chemical compounds, and it is also the underlying principle of magnetic resonance imaging and a technique used to build quantum computers.

5.5.1. NMR Experiment Overview

In general, NMR experiment starts with a preparation of the sample. In chemistry, most NMR spectra are recorded for compounds dissolved in a solvent. Therefore, signals can be observed for the solvent as well as for the compound and this must be accounted for in solving spectral problems. To avoid spectra dominated by the solvent signal, most spectra are recorded in a depurated solvent like chloroform (CDCl3), acetone-d6, D_2O, or benzene-d6. Prepared sample is held by an NMR probe. This device is placed into the core of the magnet. The probe contains also the coil for irradiating the sample with radio frequency energy and for receiving the very weak radio frequency resonance back from the sample.

NMR experiment is a very complex process. While running experiment on NMR spectrometer, the FID (continuous NMR radio-frequency) signal is registered, which is in turn a function of electromagnetic radiation absorption in time domain. This signal is then converted to the numeric value.

In general, an NMR spectrometer consists of a superconducting magnet, NMR console and a computer workstation with the control software. Thus, the spectrometer provides a uniform, stable magnetic field generated by the magnet (commonly a superconducting solenoid), creates radiofrequency (RF) pulses at the proper frequency (NMR console generates them), collects and processes the data (computer's part of the job).

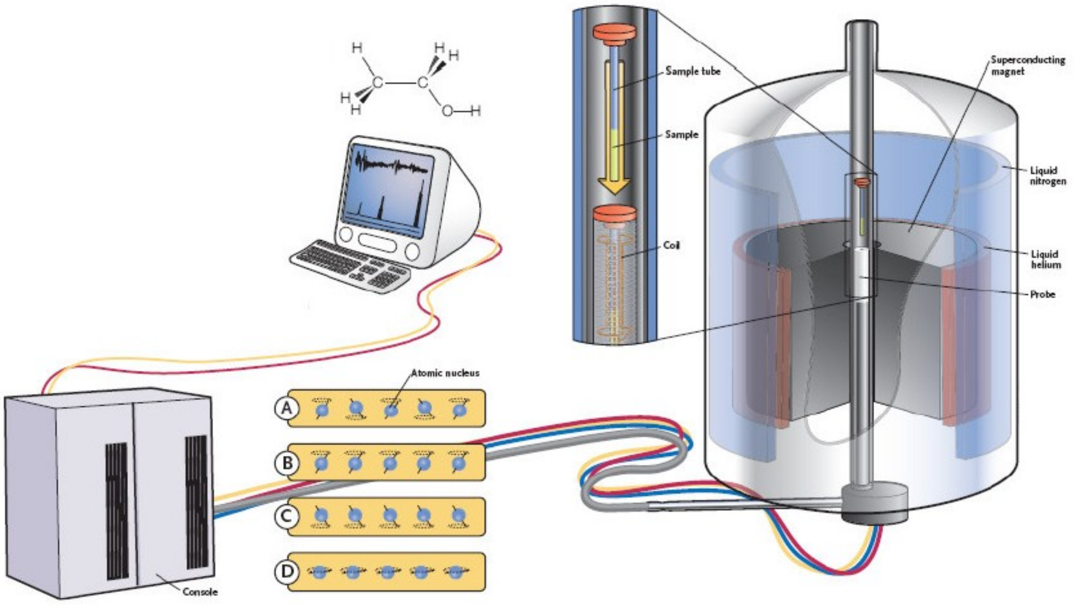


Figure 5.5.1: NMR spectrometer schema (Perkel, 2004).

Information stored in the NMR spectra received in time domain is illegible. To obtain a conventional spectrum of the NMR signal in the frequency domain, the FID signal has to be converted using mathematical operations e.g. Fourier transformation (FT), which operates on functions depending on both time and frequency.

In NMR spectroscopy the Fast Fourier Transformation (FFT) algorithm is commonly used. Data obtained in time domain in NMR experiment are then numerically processed using one of many available specialized applications, such as: VNMR, Xwin-NMR, TopSpin, Felix, NMRPipe, GIFA, MestRe-C.

5.5.2. Data Rate

User requirements in terms of date rate are changeable and depend on many aspects such as substance amount, type of the experiment (1D or nD; homo- or heteronuclear), examined isotope properties (like natural abundance, magnetogyric ratio).

In general it can be said the measurement time for 1D experiment is between few up to dozen of minutes, for 2D — up to dozen of hours and for 3D experiments between dozen of hours up to few days. Within given time they generate from dozen of kilobytes up to tens or hundreds of megabytes respectively. It means that data rate should not be a critical issue.

5.5.3. Timeliness

This is a problematic issue. When the NMR user is pretty sure the experiment is going to go smoothly (which is usually known at the start point of the experiment) and the results obtained will satisfy him/her there is no need to control the experiment in the real time. In other words the user can wait until the experiment finishes even if it lasts for few hours or days and then he/she can start to analyze the results. In case the user is not sure what the results will be like or there is a need to change some parameters — in most of the cases — he/she assists the ongoing experiment and checks its current state.

The data from the NMR experiment are usually received in packs after the certain amount of — so called — scans is processed.

5.5.4. Interactivity

In some cases, when NMR experiment is being prepared (sample locking, shimming, calibrating) latency during user-device interaction may cause difficulties or even bad experiment preparation. As the result the output data time used for spectrometer usage will be lost and measurement will have to be repeated. Such delays in interaction are less important when the data (spectrums) are processed or output data are analyzed. Those delays are rather user irritating and do not cause any problems as it was described in the previous earlier.

To minimize such situations VLab provided a compressed and efficiently compressed and secured VNC connection between the user and NMR spectrometer.

Security is one of the most important aspects in the user access to the spectrometer due to high data value obtained. A general architecture is given below.

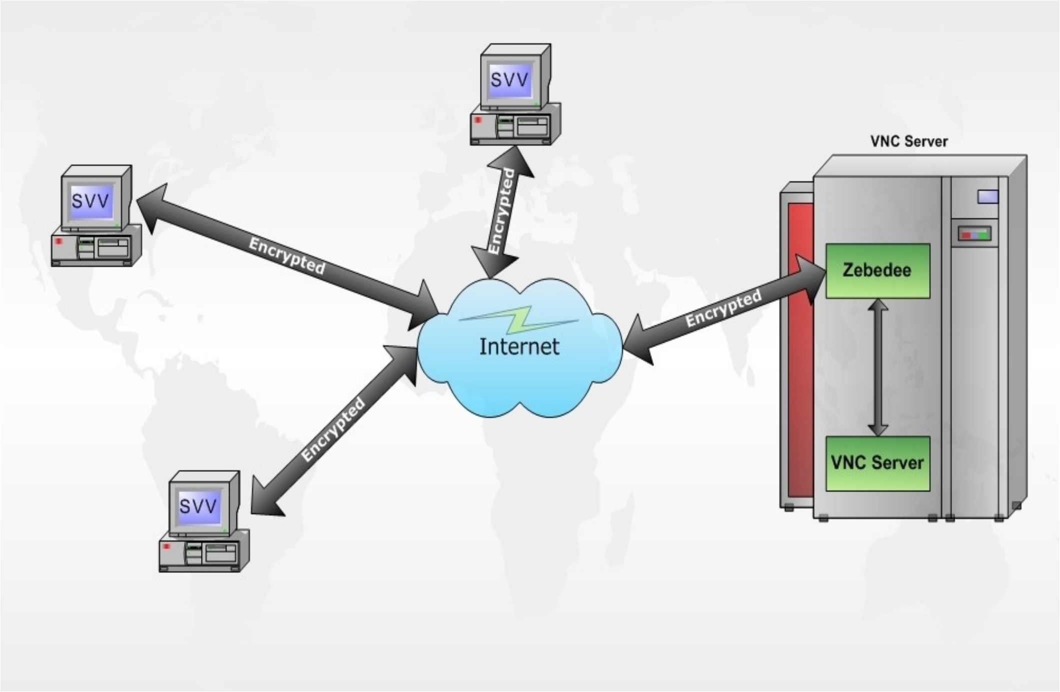


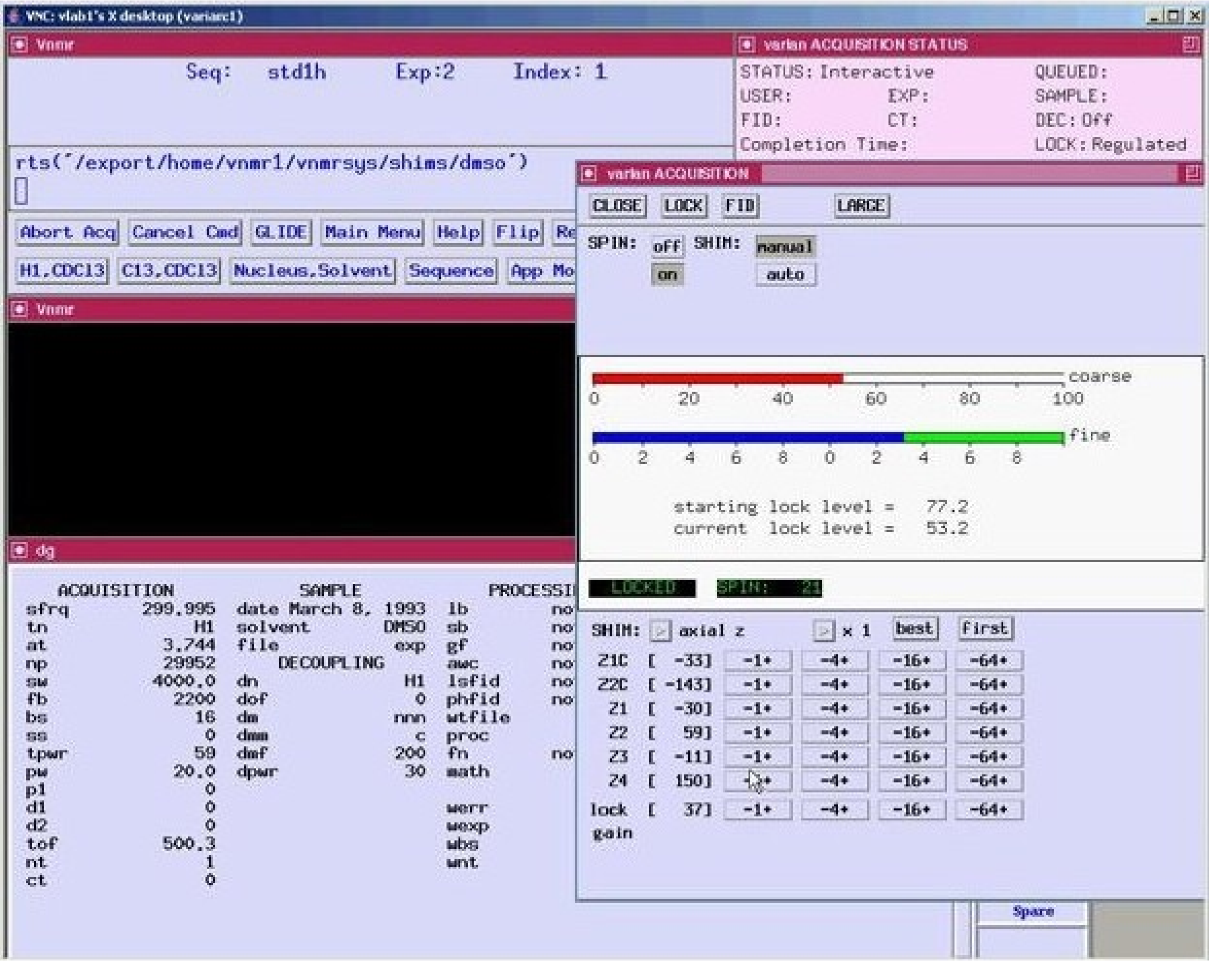
Figure 5.5.2: VLab setup for efficient communication with the NMR spectrometer.

The original VNC solution has been modified to enable secure connections between the Viewer and the Server. As you can see in the figure above there are three elements in the model:

- Secure VNC Viewer (SVV)
- Zebedee Server
- VNC Server

VLab uses the Zebedee server to encrypt the data flow. Generally speaking the Zebedee is a simple program to establish an encrypted, compressed tunnel between two systems. The encrypted data flow is sent to the Zebedee Server, which decodes the stream and forwards it to the VNC Server. The similar situation takes place when the VNC Server sends the data stream to the Viewer. First of all, the data is transmitted to the Zebedee, which encrypts the flow and sends it to the Viewer.

The Secure VNC Viewer (SVV) has been designed to decode the encrypted stream. It is a modified version of the Java VNC Viewer and has been equipped with the ability to decode the Zebedee data streams. The example screenshot is given below. The data streams carrying the visualization data are compressed and thus network throughput is not the most important issue.



```
Seq:   std1h      Exp:2      Index: 1
```

```
rts('/export/home/vnmr1/vnmrsys/shims/dmso')
```

```
STATUS: Interactive              QUEUED:
USER:            EXP:            SAMPLE:
FID:             CT:             DEC: Off
Completion Time:                 LOCK: Regulated
```

```
SPIN: off  SHIM: manual
      on         auto
```

```
starting lock level =   77.2
current  lock level =   53.2
```

```
SHIM:  axial z     x 1
 Z1C  [  -33]
 Z2C  [ -143]
  Z1  [  -30]
  Z2  [   59]
  Z3  [  -11]
  Z4  [  150]
lock  [   37]
gain
```

```
   ACQUISITION             SAMPLE                PROCESSI
sfrq     299.995  date March 8, 1993  lb            no
tn            H1  solvent       DMSO  sb            no
at         3.744  file           exp  gf            no
np         29952      DECOUPLING      awc           no
sw        4000.0  dn              H1  lsfid         no
fb          2200  dof              0  phfid         no
bs            16  dm             nnn  wtfile
ss             0  dmm              c  proc
tpwr          59  dmf            200  fn            no
pw          20.0  dpwr            30  math
p1             0
d1             0                      werr
d2             0                      wexp
tof        500.3                      wbs
nt             1                      wnt
ct             0
```

Figure 5.5.3: VLab SVV screenshot.

5.5.5. Data Loss

Another important issue that should be considered in NMR is the possibility of losing the data or some part of the data during the experiment. In case the sudden lack of the electricity or when the computer which is connected to the spectrometer will be shutdown then some parts of the data collected can be restored because every few or dozen of minutes data are stored on the hard disk of the computer steering the spectrometer. It has to be underlined that — in theory — the possibility exists to resume the experiment which has been prematurely interrupted. In case when (for example) something happens to the sample then false data will be stored in the file (additionally to the good ones saved earlier) and the experiment will have to be repeated.

5.5.6. Storage Space

Storage space required for storing NMR experiments results strongly depends on the experiment parameters, such as spectrum resolution, dimensions (1D, 2D, 3D, etc.). As mentioned before as the result of NMR experiment FID signal is registered. For 1D experiment one FID signal is obtained (dozen of kBs, with processed data file size up to 0.5 MB), but in nD experiment the number of FID series decides what resolution of the spectra in full resolution will be like. For example, in 2D experiment with resolution 2048x512 points we receive collection of 512 FIDs, each one with the resolution of 2048 points, so we will need 512 times bigger disk space than in 1D experiment. For the 3D experiment with the resolution of 1024x128x128 we receive collection of 16384 (128x128) FIDs and thus much bigger disk space is required. It is also worth to mention that 4D and 5D experiments are also ran but because they are time consuming they are rarely used.

Storage space should not be an issue for the user and it should be hidden from the user as much transparently as possible.

Taking advantage of running experiments in the VLab environment make it easy to manage experiments result files transfer between applications used in NMR spectroscopy for pre-processing, experiments and post-processing as well and keep everything together. Files are saved in Data Management System repository which is available for VLab users and where they can share results of their work. Files are copied between user experiment result files location and DMS repository using GridFTP protocol.

5.5.7. Collaboration Tools

The following are the key issues that should be taken into consideration within collaboration tools in NMR:

- Getting in touch with other people working on the same topics using collaboration tools such as: chat, audio and video communicators;
- Communicating with the device control staff. It is especially helpful when user requires the information about device time availability;
- Setting up the experiment execution time (reservation) — useful in case of on-line experiments when scientists are able to meet at the certain amount of time and watch the experiment results on the screen;

VLab enables the user with the tools such as Skype or Gadu-Gadu (the latter being a very popular Polish messenger).

5.5.8. References

[1] VLab homepage http://vlab.psnc.pl

[2] Zebedee manual, http://www.winton.org.uk/zebedee/manual.html

[3] RealVNC home page, http://www.realvnc.com

[4] Perkel, J.M., (2004) Technology. How it works. Nuclear Magnetic Resonance Spectrometer. The Scientist September 13, 32-33

6. Use Cases

6.1. Southern Astrophysical Research Telescope Control System

The SOAR (SOuthern Astrophysical Research) Telescope is designed to carry a large instrument payload. An Instrument Support Box (ISB) at each Nasmyth focus can carry a cluster of three instruments with a total weight of up to 3000kg, and contains a shared Tip-Tilt guider and calibration unit. Two "Folded Cassegrain" ports on the elevation ring can each support an additional smaller instrument weighing up to 300kg. A third such port holds the Calibration Wavefront Sensor used to tune the Active Optical System. The system is designed to allow the observer to switch between instruments, several of which will be "science ready" at any time, within a few minutes. The tertiary mirror rotates to select the focal station in use while beam steering optics within each ISB direct the light to the chosen instrument.

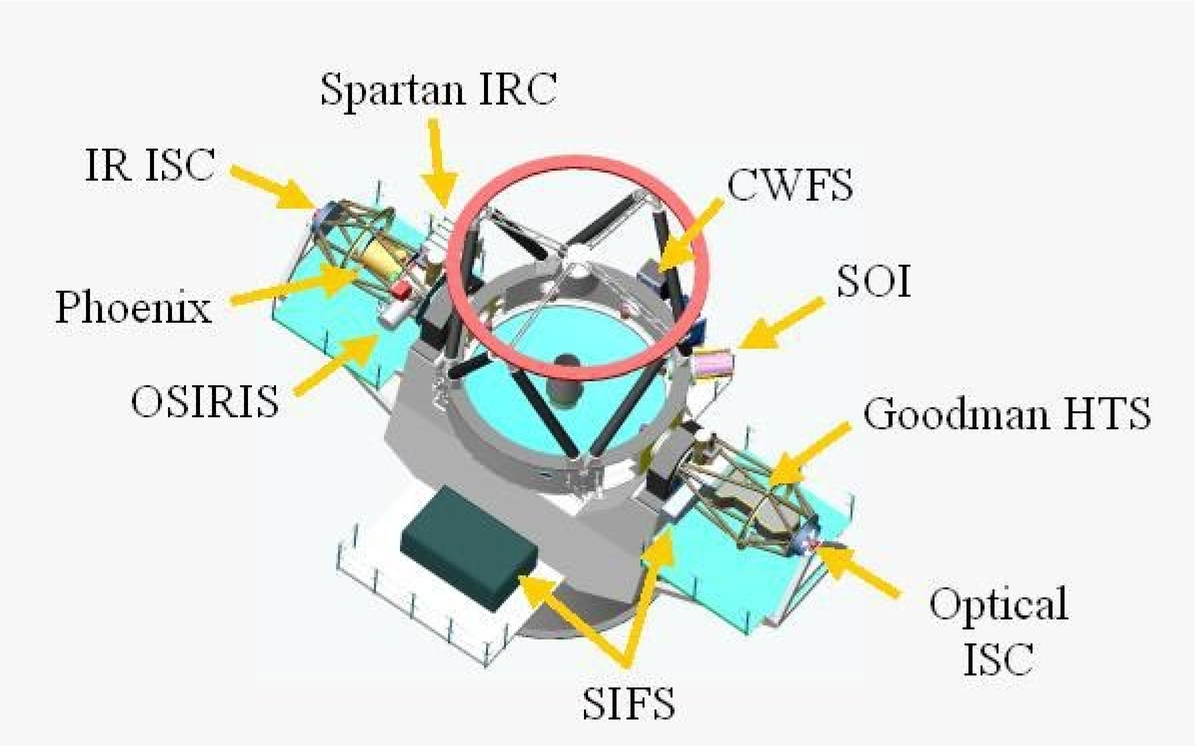


Figure 6.1.1: Schematic representation of the SOAR telescope.

The 4.1 meter SOuthern Astrophysical Research (SOAR) Telescope is now entering the operations phase, after a period of construction and system commissioning. The SOAR TCS (Telescope Control System), implemented in the LabVIEW software package, has kept pace throughout development with the installation of the other telescope subsystems, and has proven to be a key component for the successful deployment of SOAR. In order to understand the success of the SOAR TCS, we review the design considerations and the implementations details, followed by a presentation of the software extensions that allows a seamless integration of instruments into the system, as well as the programming techniques that permit the execution of remote observing procedures.

6.1.1. Introduction

With the arrival of the optics system in January 2004 the SOAR telescope is now in the commissioning stage with all subsystems fully operational and approaching readiness for science operations.

The SOAR TCS is a distributed control system, implemented using the LabVIEW software package and makes use of several support software modules written in C/C++ and TCL. The TCS software runs on regular desktop PCs with a standard set of I/O cards and peripherals. This implementation of the TCS with its strong emphasis on connectivity to diverse systems, based on a simple client/server architecture adopted by all the subsystems, played a key role in the integration of all telescope components.The extensive set of functions and visual tools offered by LabVIEW also allowed the implementation of rich user interfaces, that readily evolve in line with changing needs, and are easily tuned to optimize machine operator interactions. These capabilities have been utilized at all levels during the commissioning period, both for quick visualization of status and data, and to develop control sequences and ad-hoc operations.

6.1.2. Integration Results

The first outstanding result is the open loop tracking performance of the mount. The mount tracks very well with no perceivable drift of the tracked object for several minutes. The good tracking performance proved to be important during the calibration of the optics. The SOAR Calibration WaveFront Sensor (CWFS) used to measure the aberrations, has no provisions for guiding so that it is necessary to center a star on the 3 arcsecond diameter entrance aperture of the device and hold it there during a series of exposures ranging from 10 to 30 seconds while tracking open loop. Without the good tracking this would have been very time consuming, or impossible activity.

The implication of the good open loop tracking performance, is that the computation of the demands is done at the right time and with the right precision, and that the mount follows those demands with great accuracy (see ref 3 for a detailed description of the mount results). In order to better understand how the demands are generated in the SOAR TCS, figure 6.1.2 shows the present implementation of the KERNEL5,6 component and the associated support software that performs the demand computations.

The KERNEL running under the Real-Time Applications Interface extension to Linux (RTAI-Linux), computes the position demands for the mount, rotators and guiders, and sends those demands to the mount and the LabVIEW process for final application of the results. An RTAI-FIFO link is used to expedite the KERNEL to LabVIEW communications in these time-critical cases, because TCP/IP used for the original implementation, was found to impose an unacceptable overhead. Conversely, TCP/IP is entirely adequate for the TCL link that implements less time critical services, such as slew requests, pointing measurements, etc., and given its proven reliability there is no need to move them to the FIFO link.

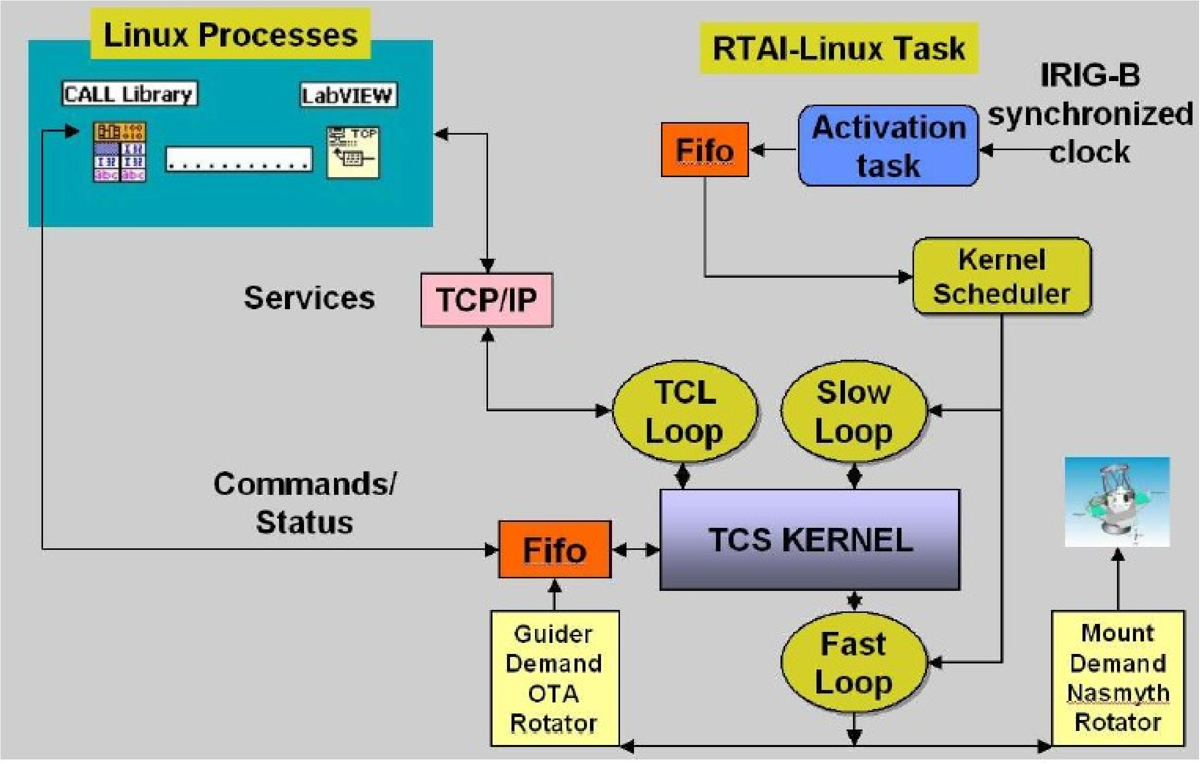
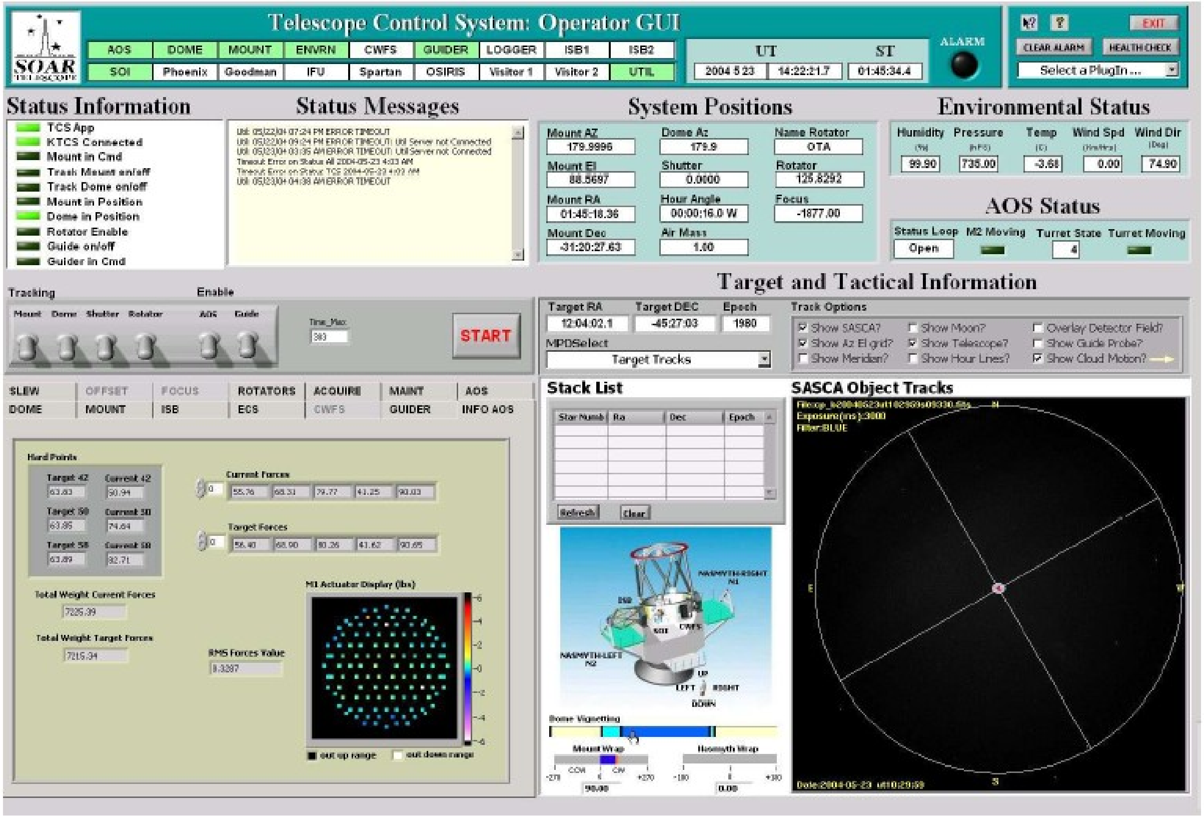


Figure 6.1.2: KERNEL component and associated support software.

The second important result is the stability of the messaging system. The communications library provides applications with a standard messaging interface supporting a client/server architecture. Furthermore, SOAR defines a messaging API using a tokenized scheme. Underlying this architecture is the need to modularize applications , with many modules running concurrently exchanging messages with one another. Messages may be exchanged between subsystems in different machines, as well as internal components running in the same computer. Different transport mechanisms are used, like TCP/IP, RTAI-FIFOS, queues, shared memory, serial links, etc. The ability to use diverse transports for messages proved to be extremely useful when performance enters the picture. Whenever it was felt that processes were lagging behind because of the slowness in message exchanges, a new faster transport was explored. A typical result is that the KERNEL communications could be upgraded from the pure TCP/IP transport to RTAI-FIFO, regaining the lost performance without changing the architecture or affecting the stability of the system.

Another important consideration for a client/server architecture to be successful is to design the messaging system to be non blocking and for servers to decouple the internal processing activities from the handling of the messages. On the one hand, a server should respond immediately to every command, while on the other, the act of responding to commands must not interfere with the control activities. This results in the ability to decompose a software problem into a series of fully encapsulated processes and to keep adding new processes without disrupting other components in the system, i.e., so that the application can evolve and change under control. The practical result is that the SOAR TCS is a very stable system, running for days without a flaw, and behaving gracefully when a component does fail, continuing to function with degraded performance, but remaining responsive to external demands.

Another area of satisfactory results concerns operator-TCS interactions. SOAR utilizes GUI based interfaces at all levels, from basic motor control all the way to elaborate maneuvers like finding targets and setting the optics. Figure 6.1.3 presents a snapshot of the present operator GUI.



UT 2004 5 23 14:22:21.7 ST 01:45:34.4

Util: 05/22/04 07:24 PM ERROR TIMEOUT
Util: 05/22/04 09:24 PM ERROR TIMEOUT: Util Server not Connected
Util: 05/23/04 03:35 AM ERROR TIMEOUT: Util Server not Connected
Timeout Error on Status All 2004-05-23 4:03 AM
Timeout Error on Status TCS 2004-05-23 4:03 AM
Util: 05/23/04 04:38 AM ERROR TIMEOUT

Mount AZ 179.9996
Mount El 88.5697
Mount RA 01:45:18.36
Mount Dec -31:20:27.63
Dome Az 179.9
Shutter 0.0000
Hour Angle 00:00:16.0 W
Air Mass 1.00
Name Rotator OTA
Rotator 125.8292
Focus -1877.00

Humidity (%) 99.90
Pressure (hPa) 735.00
Temp (C) -3.68
Wind Spd (Km/Hrs) 0.00
Wind Dir (Deg) 74.90

Status Loop Open
M2 Moving
Turret State 4
Turret Moving

Time_Max: 300

Target RA 12:04:02.1
Target DEC -45:27:03
Epoch 1980
MPDSelect Target Tracks

Track Options
☑ Show SASCA?
☑ Show Az El grid?
☐ Show Meridian?
☐ Show Moon?
☑ Show Telescope?
☐ Show Hour Lines?
☐ Overlay Detector Field?
☐ Show Guide Probe?
☑ Show Cloud Motion?

Target 42 63.83
Current 42 50.94
Target 50 63.85
Current 50 74.64
Target 58 63.89
Current 58 82.71

Current Forces 0 55.76 68.31 79.77 41.25 90.03
Target Forces 0 56.40 68.90 80.26 41.62 90.65

Total Weight Current Forces 7225.39
Total Weight Target Forces 7215.34
RMS Forces Value 0.3287

Mount Wrap 90.00
Nasmyth Wrap 0.00

File:op_b20040523ut102939s09330.fits
Exposure(ms):3000
Filter:BLUE
Date:2004-05-23 ut10:29:59

Figure 6.1.3: SOAR operator GUI.

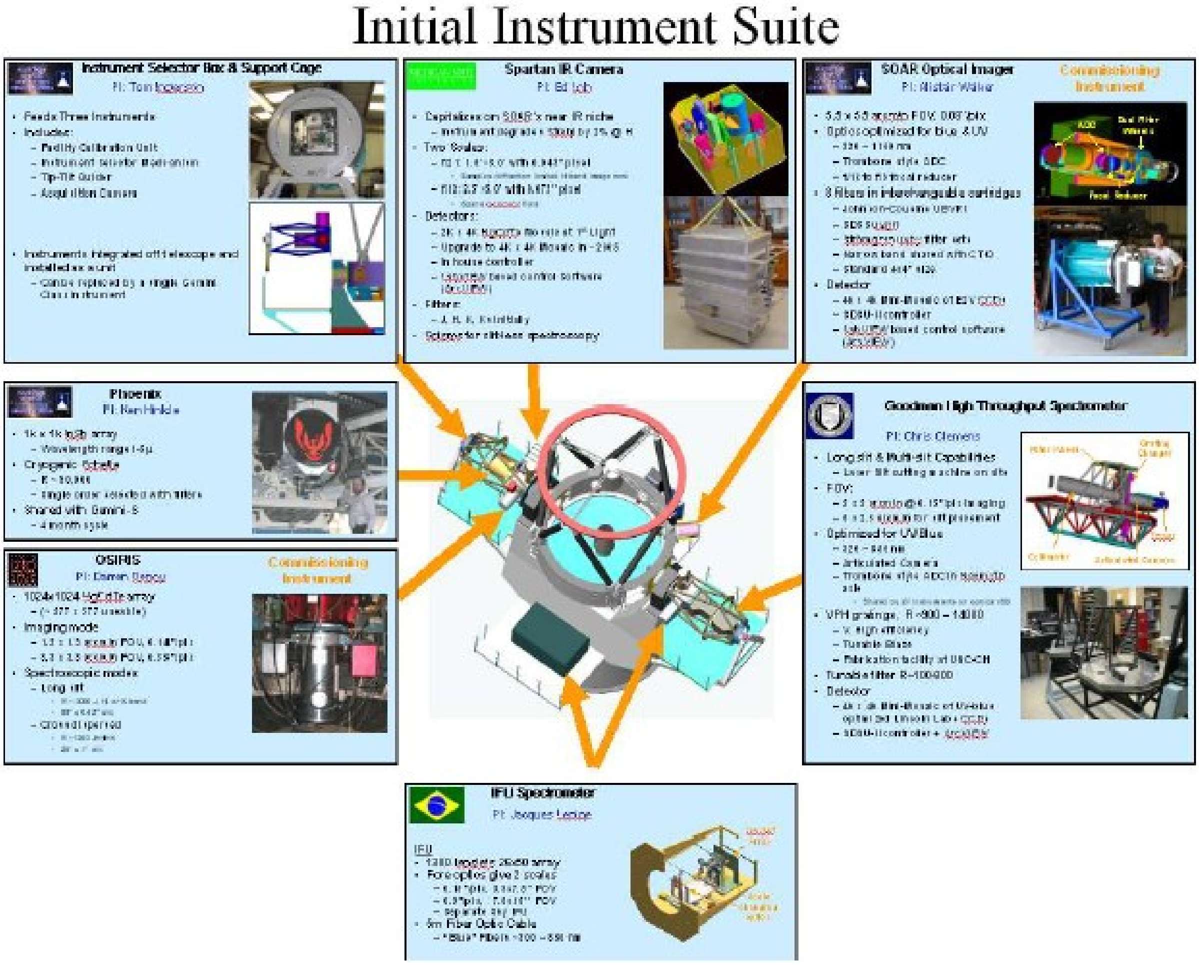
The critical aspect here is the rational use of the screen space, while avoiding overloading the user with information. The general idea is that at any moment only the minimum amount of information should be presented to give an indication of the system status. For example, the two upper top left rows of indicators represent the status of all the subsystems and instruments connected. A colour code serves to provide status information, anomalous conditions being flagged by changes of indicator colour. Clicking the trouble indicator pops up a status message with a brief description of the problem. The operator may then choose to open the entire screen of the subsystem involved in order to perform a detailed diagnosis of the problem. We utilize a dual screen system, in which such detailed sub-system display are diverted to a second monitor so as not to overlap the operatorGUI and hiding the overview of the rest of the system.

The bottom left section is where all actions are initiated. This is a "tab" structure, where one can quickly select a section and activate the required actions. The idea here is to never be more that a click away from any operation, a condition difficult to attain with more traditional menu approaches. Some common operations, like track on/off, dome enable/disable, etc., are promoted to an actual switch icon as shown in the middle row of on/off switches. The centre section to the right of the screen contains a number of numeric indicators which give the present position of the mount, dome, and rotator, as well as the environmental conditions. In the bottom right section there is an image generated by the Soar All Sky Camera (SASCA) showing the state of the sky at the time of observation.

6.1.3. Instruments Integration

The SOAR telescope will deploy multiple "hot" instruments when in operations. Figure 6.1.4 shows the initial suite of instruments, and the locations where they will be installed on the telescope. At the Nasmyth locations, instruments attach to a support cage known as the Instrument Selector Box (ISB), which contains movable optics to send the light to any of the three instruments installed at each side. A guider device is also included in each ISB. Selection of the instrument to be used, consists of rotating the M3 turret to the appropriate port and setting the optics in the ISB if necessary.

Given the number of online instruments and the ability to rapidly switch from one to another, the SOAR telescope is well suited for the study of short lived transient objects, such as gamma ray bursts, by means of a "target of opportunity program".



Initial Instrument Suite

Instrument Selector Box & Support Cage
PI: Tom Ingerson
- Feeds Three Instruments
- Includes:
  - Facility Calibration Unit
  - Instrument Selector Mechanism
  - Tip-Tilt Guider
  - Acquisition Camera
- Instruments integrated off telescope and installed as a unit
  - Can be replaced by a single Gemini Class instrument

Spartan IR Camera
PI: Ed Loh
- Capitalizes on SOAR's near IR niche
  - Instrument degrades [illegible] by 3% @ H
- Two Scales:
  - f/21: 1.4'×5.0' with 0.043" pixel
    - [illegible]
  - f/12: 2.5'×5.0' with 0.071" pixel
    - [illegible]
- Detectors:
  - 2K × 4K HgCdTe Mosaic at 1st Light
  - Upgrade to 4K × 4K Mosaic in ~2005
  - In house controller
  - LabVIEW based control software (ArcVIEW)
- Filters:
  - J, H, K, Kshort [illegible]
- Science for slitless spectroscopy

SOAR Optical Imager
PI: Alistair Walker
Commissioning Instrument
- 5.5 x 5.5 arcmin FOV, 0.08"/pix
- Optics optimized for blue & UV
  - 320 – 1100 nm
  - Trombone style ADC
  - f/16 to f/9 focal reducer
- 8 filters in interchangeable cartridges
  - Johnson-Cousins UBVRI
  - SDSS u'g'r'i'z'
  - Strömgren uvby filter sets
  - Narrow band shared with CTIO
  - Standard 4x4" size
- Detector
  - 4K × 4K Mini-Mosaic of E2V CCDs
  - SDSU-II controller
  - LabVIEW based control software (ArcVIEW)

Phoenix
PI: Ken Hinkle
- 1k x 1k InSb array
  - Wavelength range 1-5μ
- Cryogenic Echelle
  - R ~ 50,000
  - Single order selected with filters
- Shared with Gemini-S
  - 4 month cycle

OSIRIS
PI: Darren Depoy
Commissioning Instrument
- 1024x1024 HgCdTe array
  - (~ 577 × 577 useable)
- Imaging mode
  - 1.3 × 1.3 arcmin FOV, 0.14"/pix
  - 3.3 × 3.3 arcmin FOV, 0.35"/pix
- Spectroscopic modes
  - Long slit
    - [illegible]
    - [illegible]
  - Cross dispersed
    - [illegible]
    - [illegible]

Goodman High Throughput Spectrometer
PI: Chris Clemens
- Long slit & Multi-slit Capabilities
  - Laser slit cutting machine on site
- FOV:
  - 5 × 5 arcmin @ 0.15"/pix imaging
  - 4 × 3.3 arcmin for slit placement
- Optimized for UV/Blue
  - 320 – 850 nm
  - Articulated Camera
  - Trombone style ADC in SOAR ISB [illegible]
    - [illegible]
- VPH gratings, R ~900 – 14000
  - V. high efficiency
  - Tunable Blaze
  - Fabrication facility at UNC-CH
- Tunable filter R~100-900
- Detector
  - 4K × 4K Mini-Mosaic of UV-blue optimized Lincoln Lab CCDs
  - SDSU-II controller + ArcVIEW

IFU Spectrometer
PI: Jacques Lepine
IFU
- 1300 lenslets 26x50 array
- Fore optics give 2 scales
  - 0.1"/pix, [illegible] FOV
  - 0.3"/pix, 7.8x15" FOV
  - Separate sky IFU
- 5m Fiber Optic Cable
  - "Blue" Fibers ~300 – 850 nm

Figure 6.1.4: SOAR instruments.

All the new instruments developed by the SOAR community utilizes LabVIEW to implement the control and acquisition software. The first instrument to be integrated was the SOAR Optical Imager, that is being used to commission the telescope, and includes its own rotator, ADC corrector and guider since it is located at one of the Bent Cass ports. This instrument requests status information from the TCS, and steers the focus device, when performing focus sequences. The TCS in turn steers the rotator, ADC and guider. The command exchanges follows the protocol explained earlier. A TCP/IP link provides enough bandwidth for all these operations.

The second instrument to be integrated was OSIRIS, a legacy InfraRed Imager/Spectrograph provided by Ohio State University. The software for this instrument is not LAbVIEW based, but we faced no difficulties in connecting it to the TCS given the simple client/server approach of exchanging commands. This instrument has a more demanding interaction with the TCS given the continuous requests for mount offset motions, that are required for observations in the near IR.

6.1.4. Remote Observation

One of the requirements of the SOAR TCS is the need to support remote observation. Note that the term is "remote observation" and not "remote operation", since there will always be an operator on site to enable telescope slew and to supervise other potentially hazardous maneuvers. In this context then, remote observation is the ability to interact with the TCS and instruments to get observational data from a remote location, with the assistance of a local operator.

Traditionally, this interaction has been made by remote display of screens generated at the telescope site. The drawback in this situation, is that response times are not optimum, which can be irritating, and to improve on that condition the bandwidth requirements increase. Our approach (see ref 4 for a complete description) makes use of three technologies to accomplish a better solution: DataSocket, Remote Panels and Compression.

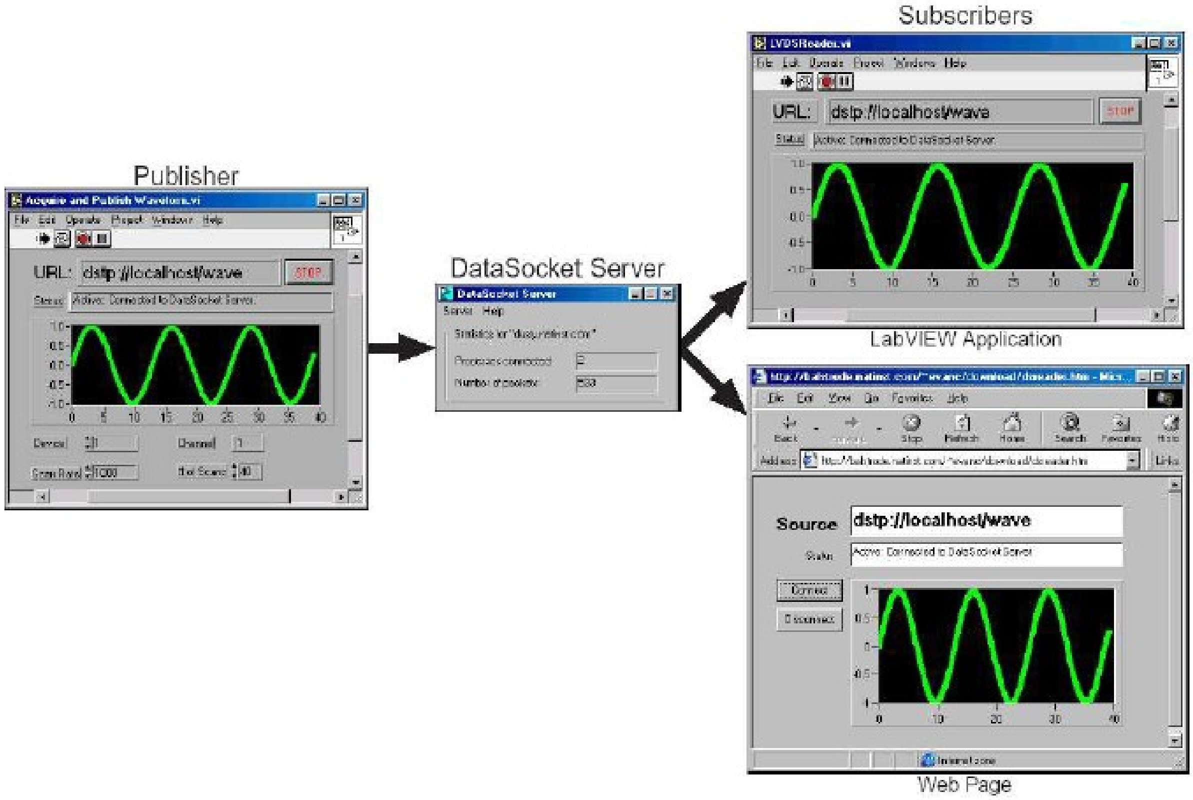

Subscribers
LVDSReader.vi
URL: dstp://localhost/wave
Publisher
Acquire and Publish Waveform.vi
URL: dstp://localhost/wave
STOP
DataSocket Server
LabVIEW Application
Source dstp://localhost/wave
Connect
Disconnect
Web Page

Figure 6.1.5: DataSocket concept.

DataSocket (http://ni.com/datasocket) is a programming technology for sharing and publishing live data between different applications. Broadcasting data requires three "actors": a publisher, the DataSocket Server, and a subscriber (see figure 6.1.5). A publishing application uses the DataSocket API to write data to the server. A subscribing application uses the DataSocket API to read data from the server. The server then broadcast information to subscribers whenever a producer generates new data. The actors normally reside on different machines to improve performance. We use DataSocket to publish telemetry data from the TCS and instruments. The remote subscribers generate local display screens or Web pages, to visualize the data, thus decreasing the bandwidth requirements considerably.

Remote Panels in the context of LabVIEW is the ability to view or control a VI in any Web browser. This feature allows several users at different locations to access the VI simultaneously. What makes this approach different to previous remote display implementations, is that the amount of data exchange between the machine running the VI and the machines displaying the VI is greatly reduced, by sending only the portion of the display that have changed (if any), without the need to resend the entire panel. This is done with the help of a plug-in (freely available from ni.com) that one needs to install in a browser, with this plug-in performing the display updates locally.

6.1.5. Summary

After 5 years of development and test, the SOAR TCS software is in a stable condition, and utilized routinely to control the SOAR telescope and instruments. The software survived several LabVIEW versions (from 5.1 to 7.0) and several Linux versions (RedHat 5 to RedHat 9) during the implementation phase. Upgrading to new versions has been seamless and straightforward. We expect this trend to continue as new releases are produced.

6.2. The National Virtual Observatory Project and the International Virtual Observatory Alliance

For several centuries, astronomical research has been carried out by a single astronomer or small group of astronomers performing observations of a small number of objects. In the past, entire careers have been spent in the acquisition of enough data to barely enable statistically significant conclusions to be drawn. Moreover, because observing time with the most powerful facilities is very limited, many astrophysical questions that require a large amount of data for their resolution simply could not be addressed.

This approach is now undergoing a dramatic and very rapid change. The transformation is being driven by the unprecedented technological developments over the last decade. The major areas of change upon which this revolution in astronomy rests are advances in telescope design and fabrication, the development of large-scale detector arrays, the exponential growth of computing capability, an increasing capability of space-based observatories and missions, and the ever-expanding coverage and capacity of communications networks.

The steep increase in the volume and complexity of available information is based on the great progress in technology, including digital imaging (the chief data source in astronomy), and, of course, the ways of processing, storing, and accessing information with the advent of grid technology [7]. Figure 6.2.1 shows, as an example, the Montage project [10] architecture, on how data grid technology can be applied to manage astronomy objects creation or replica retrieval, by the use of data grid technology. The Montage project goal is to deploy a portable, compute-intensive service that will deliver science-grade custom mosaics on demand, with requests made through existing portals. Science-grade in this context requires that terrestrial and instrumental features are removed from images in a way that can be described quantitatively; custom refers to user-specified parameters of projection, coordinates, size, rotation and spatial sampling.

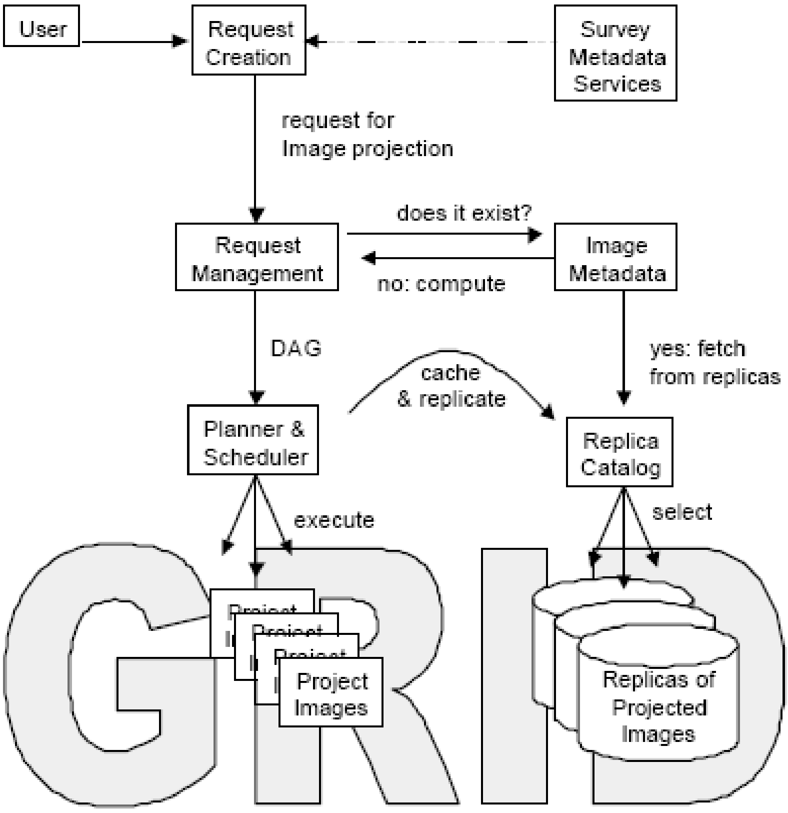


Figure 6.2.1: The MONTAGE project architecture.

The diagram exemplifies the use of grid technology in the VO. After a user request has been created and sent to the Request Manager, part of the request may be satisfied from existing (cached) data. The Image Metadata (IM) system looks for a suitable file, and if found, gets it from the distributed Replica Catalog (RC). If not found, a suitable computational graph (DAG) is assembled and sent to be executed on grid resources. Resulting products may be registered with the IM and stored in RC. The user is notified that the requested data is available.

Most of the scientific measurements and data obtained today are either generated in a digital form or converted to it. Most of all astronomical measurements today are digital in nature, and most instruments contain some form of a digital imaging arrays. Such devices, in turn, are based on the same technology (integrated circuits and microelectronics), governed by Moore's law and are thus growing exponentially in their information-generating ability.

In addition to this increased data rate, the manner in which observations are being made is also changing. Although the new observatories in space and on the ground still devote a significant fraction of their time to research in the single observer/single program mode where small blocks of time are allocated to many specifically targeted research programs, more time is now being devoted to large scale surveys of the sky, often at multiple wavelengths, that involve large numbers of collaborators. These large survey programs will produce coherent blocks of data obtained with uniform standards and with the amount of data often measured in terabytes.

These trends will continue. For example, the Large-Aperture Synoptic Survey Telescope (LSST) [11] should produce about 10 terabytes per day.

The astronomical community should take advantage of the advances in computational speed, storage media and detector technology in two ways:

1. By carrying out new generation surveys spanning a wide range of wavelengths and optimized to exploit these advances fully;
2. By developing the software tools to enable discovery of new patterns in the multi-Terabyte (and later Petabyte) databases that represent the legacies ofthese surveys.

In combination, new generation surveys and software tools can provide the basis for enabling science of a qualitatively different nature.

The quantity of data continues to grow exponentially. Transforming vast masses of bits into refined knowledge and understanding of the universe is a highly complex task. The great quantitative change in the amount and complexity of available scientific information should lead to a qualitative change in the way science is conducted.

Large digital sky surveys and archives are now becoming the principal sources of data for astronomy. Increasingly, the field is being dominated by the analysis of large, uniform sky surveys, sampling millions or billions of sources, and providing tens or hundreds of measured attributes for each of them. There is a paradigm shift in observational astronomy, with survey based science becoming an ever more important way of exploring the universe in a systematic way. The sky is now being surveyed over a full range of wavelengths, giving us, at least in principle, a panchromatic and less biased view of the universe.

The tools to carry out surveys over nearly the entire electromagnetic spectrum on a variety of spatial scales and over multiple epochs are now available, all with well-defined selection criteria and well-understood limits. The ability to create panchromatic images, and in some cases digital movies of the universe, provide unprecedented opportunities for discovering new phenomena and patterns that can fundamentally alter our understanding. In the past, a panchromatic view of the same region of sky at optical and radio wavelengths led to the discovery of quasars. The availability of infrared data led to the discovery of obscured active galactic nuclei and star forming regions unsuspected from visible images. Repeated images of the sky have led to the discovery of transient phenomena — supernovae, and more recently, micro-lensing events — as well as a deeper understanding of synoptic phenomena. The joining together of various largescale digital surveys will make possible new explorations of parameter space, such as the low surface brightness universe at all wavelengths.

Many astronomical surveys, large telescopes, and space missions are already producing large quantities of high quality legacy data, and much of this is currently being archived. Most of these data, obtained through use of costly and highly oversubscribed, state of the art facilities, have an unprecedented richness and depth, and they offer unique opportunities for application to a variety of scientific programs by a wide range of users. The existence of such information-rich archives, containing multi-wavelength data on hundreds of millions of objects, is creating a demand within the astronomical community for access to the archives and for the tools necessary to analyze the data they contain. Opportunities for data mining, for sophisticated pattern recognition, for large scale statistical cross correlations, and for the discovery of rare objects and temporal variations all become apparent.

In addition, for the first time in the history of astronomy, such data sets can go through meaningful comparisons to be made between sophisticated numerical simulations and statistically complete multivariate bodies of data. The rapid growth of high speed and widely distributed networks means that all of these scientific endeavours will be made available to the community of astronomers throughout the U.S. and in other countries.

The potential for scientific discovery afforded by these new surveys is enormous. Entirely new and unexpected scientific results of major significance will emerge from the combined use of the resulting datasets, science that would not be possible from such sets used on their own.

6.2.1. The US-NVO

The NVO is the National Virtual Observatory, the United States based Virtual Observatory [8] project that is collaborating with the International Virtual Observatory Alliance (IVOA) to make it possible for astronomical researchers to find, retrieve, and analyse astronomical data from ground and space-based telescopes worldwide.

The origin of the NVO can be traced to the establishment in the early 1990s of wavelength-oriented science archive centres for NASA mission datasets. These were the first comprehensive astronomy archive facilities having a close connection between data and expertise in calibrating and using the data. Also, during the 1990s several large-scale digital sky surveys began, most notably the Sloan and 2MASS surveys. The images and source catalogues derived from these surveys demonstrated the value of homogeneous, on-line datasets. In April 1999, the concept for a "National Virtual Observatory" arose at a meeting of the Decadal Survey Panel on Theory, Computation, and Data Discovery. In the following two years, a series of workshops and conferences were held to flesh out the concept of the VO. In September 2001, National Science Foundation (NSF) Information Technology Research program awarded US$10Million to a 17-organization collaboration led by Alex Szalay (JHU) and Paul Messina (Caltech) to build the infrastructure for the VO. Both the US NVO project and the Astrophysical Virtual Observatory, the European pilot VO effort, released their first science prototypes in January 2003.

The VO enables a new way of doing astronomy, moving from an era of observations of small, carefully selected samples of objects in one or a few wavelength bands, to the use of multi-wavelength data for millions, if not billions of objects. Such datasets will allow researchers to discover subtle but significant patterns in statistically rich and unbiased databases, and to understand complex astrophysical systems through the comparison of data to numerical simulations. The VO goal is to provide simultaneous access to multi-wavelength archives and advanced visualization and statistical analysis tools.

The Virtual Observatory comes about now as a result of the convergence of research interests (multi-wavelength astrophysics, archival research, survey astronomy, and temporal astronomy) and information technology (Moore's law, digital detectors, the Internet, and data representation standards). Astronomy is well-positioned to exploit the IT revolution because of its early commitment to formatting standards (FITS), the now universal use of digital detectors, and an ever-broadening commitment to data preservation and data re-use.

The US NVO project is supported by the National Science Foundation's Information Technology Research Program under Cooperative Agreement AST-0122449 with The Johns Hopkins University.

The NVO should not be viewed just as a new information infrastructure for data-rich astronomy. Rather, its main objective is to be a comprehensive research environment for the new astronomy with massive data sets, including data, tools, and services. The NVO goes beyond the existing structures in that it would provide new and increasingly needed technical and scientific functions, including unprecedented data fusion and data mining capabilities, instead of just the passive serving of limited data sets of the kind have been available until recently. It will make possible rapid querying of individual terabyte archives by thousands of researchers, enable visualization of multivariate patterns embedded in large catalogue and image databases, enhance discovery of complex patterns or rare phenomena, encourage real time collaborations among multiple research groups, and allow large statistical studies that will for the first time permit confrontation between databases and sophisticated numerical simulations. It will also facilitate the understanding of many of the astrophysical processes that determine the evolution of the Universe. "It will enable new science, better science, and more cost effective science" [6].

6.2.2. IVOA

The International Virtual Observatory Alliance (IVOA) [9] was formed in June 2002 with a mission to facilitate the international coordination and collaboration necessary for the development and deployment of the tools, systems and organizational structures necessary to enable the international utilization of astronomical archives as an integrated and interoperating virtual observatory. By January 2005, the IVOA has grown to include 15 funded VO (virtual observatory) projects from Australia, Canada, China, Europe, France, Germany, Hungary, India, Italy, Japan, Korea, Russia, Spain, the United Kingdom, and the United States. This membership is being expanded to include representation from projects constructing and planning new observatories and astronomical facilities, as well as emerging astronomical communities that seek to benefit fromthe global availability of VO facilities and technologies.

In January 2003, the IVOA Executive adopted several strategic approaches to achieving the IVOA mission. Firstly, the work of producing standards was to be modelled on the W3C process involving a cycle of Working Drafts, Proposed Recommendations, and finally Recommendations to the international community as represented by the International Astronomical Union (IAU). Secondly, Working Groups were created with cross-project membership in those areas where key interoperability standards and technologies had to be defined and agreed upon. The Working Groups representa significant commitmentfrom each of the projects to build new standards on a time scale consistent with the original IVOA Roadmap (2002-2005). Finally, the IVOA Executive emphasized the importance of annual demonstrations for the astronomical community of new and emerging VO capabilities. Demonstrations provide a forum to engage the scientific community, they provide a major, regular, and predictable milestone for development projects to meet deadlines, and they allow the IVOA Executive to assess progress on standards development in order to set priorities for standards roll-out in a coordinated way.

Currently the IVOA coordinates eight Working Groups as well as four Interest Groups in areas such as Applications and Theory. The efforts of the Working Groups have been coordinated and focused through five international Interoperability Workshops held in the US, the UK, France, India, and Japan between October 2002 and May 2005. To date, the Working Groups have produced Working Drafts, Technical Notes and specific Recommendations in the areas of XML data format standards (VOTable), VO Resource Metadata, Universal Content Descriptions, Space-Time Coordinate Metadata, unified Data Access Layer standards for spectra and images, VO Resource Registries, VO Query Language, unified astronomical data models and web service technologies for the VO. The chairs of the Working Groups have also produced an overall architectural plan for an operational VO that identifies the critical areas for current and future development of standards and technologies. The IVOA has also sponsored two regional VO meetings in China and India to ensure that small VO projects in these areas have a forum to share developments with each other and the larger projects in Europe and the US. In January 2003, 2004, and 2005, coordinated demonstrations of VO developments were held in the US and Europe which highlighted progressively more complex VO capabilities, new web-accessible services for the community, and downloadable prototype software that proved capable of producing new scientific results in multi-wavelength astronomy. In July 2003, a coordinated set of demonstrations was held at the IAU General Assembly by ten VO Projects that highlighted the international scope of the VO effort.

Several new scientific finding have already been published through VO-based research. The first major discovery to be made with the Virtual Observatory was reported by the Astrophysical Virtual Observatory (currently Euro-VO) in 2004 (Padovani et al.; Astronomy & Astrophysics, 424, 545). The AVO science team discovered 31 previously undetected powerful super-massive black holes in the so-called GOODS (Great Observatories Origins Deep Survey) fields. The US Virtual Observatory (NVO) also published a paper; McGlynn et al. classified all unidentified

ROSAT WGACAT objects using VO data access methods to cross-correlate multi-wavelength catalogues (Astrophysical Journal, 616, 1284, 2004). More than 400 papers have been published related to "Virtual Observatories". These publications indicate that Virtual Observatories have a very high potential to enable new astronomical research.

The IVOA has also sought to form links to other communities, projects, and governing bodies in order to share technological approaches, gather scientific and technical requirements and to promote the importance of adequate funding for the scientific exploitation of data from new and existing facilities. In particular, the IVOA has initiated the formation of an astronomical grid community research group within the Global Grid Forum, has encouraged the International Astronomical Union (IAU) Commission 5 to form a VO Working Group to facilitate IAU oversight and endorsement of IVOA recommendations, has made presentations to specific large project meetings (ALMA, LSST, and IAU Joint Discussions), and has actively participated in the OECD Global Science Forum workshops on Future Large-Scale Projects and Programmes in Astronomy and Astrophysics (April 2004). The OECD workshop summary noted: The huge volume of digital information flowing from the new observatories raises the challenges of collecting, using, storing, and sharing data. The workshops identified a number of major issues in the context of a new community-based vision for a common research infrastructure: the "Virtual Observatory". Impressive progress has been made and the momentum of the International Virtual Observatory Alliance will ensure sustained progress, provided support and funding are made available. The IVOA is also engaged in the OPTICON-sponsored study of Future Astronomical Software Environments, helping to define a framework in which legacy applications, data processing pipelines, custom analysis tools, and VO-enabled data access can be easily integrated together [9].

6.2.3. References

[6] Text based on "Towards the National Virtual Observatory: A Report Prepared by the National Virtual Observatory Science Definition Team", http://www.astro.caltech.edu/%7Egeorge/sdt/sdt-final.pdf

[7] Grids and the Virtual Observatory Roy Williams, in Grid Computing: Making The Global Infrastructure a Reality by Fran Berman, Anthony J.G. Hey, and Geoffrey Fox, Wiley, 2003, pp 837-858.

[8] US National Virtual Observatory, http://us-vo.org/index.cfm

[9] International Virtual Observatory Alliance: Roadmap, http://www.ivoa.net/pub/info/

[10] The Montage Project, http://montage.ipac.caltech.edu/

[11] The Large Synoptic Survey Telescope (LSST), http://www.lsst.org/lsst_home.shtml

6.3. Nuclear Magnetic Resonance Spectrometer

Nuclear Magnetic Resonance (NMR) spectroscopy is a unique experimental technique that is widely used in physics, organic and inorganic chemistry, biochemistry as well as in medicine.

The analysis of one- or multi-dimensional homo- and hetero-nuclear spectra obtained in the course of NMR experiment can provide information about the chemical shifts of the nuclei, scalar coupling constants, residual bipolar coupling constants and the relaxation times T1, T2.

Many areas of information can be obtained from this single phenomenon. In its simplest form NMR allows identification of individual atoms in a pure molecule. Much like using infrared spectroscopy to identify functional groups, analysis of a 1D NMR spectrum tells the scientist what atom environments (like a methyl proton), and in some cases how many atoms of each type, exist within the sample. NMR is based in quantum mechanical properties of nuclei, and as such is very reliable, predictable and reproducible. Since its advent, it has become the most important analytical tool available for organic chemists; it yields far more information than for example infrared spectroscopy.

The impact of NMR Spectroscopy on the natural sciences is substantial. It can be used to study mixtures of analytes; to understand dynamic effects such as change in temperature and reaction mechanisms; it can be used in the solution and solid state; and critically it is an invaluable tool in understanding protein and nucleic acid structure and function.

6.3.1. NMR and VLab

To explain why NMR has been introduced to the Virtual Laboratory few additional words concerning tasks types are needed.

In the Virtual Laboratory two main tasks types can be distinguished: experimental and computational ones. The latter can be divided into regular (batch) jobs and interactive/visualization tasks (the ones performed in the real time, directly by the users — via GUI). The biggest difference (and difficulty) between those types is that — in the interactive tasks — the time slot reserved for running the task on a computational machine must be synchronized with user preferences, considering specific working hours, daily schedule etc. Another aspect is the mechanism which will present the users with the graphical interface of the actual computational (or visualization) application — which is run on dynamically assigned computational server — and allow them to perform their interactive task. Another, very characteristic type of Virtual Laboratory tasks are the experiments. By the term "experiment" we mean a task, scheduled to be performed on the remote laboratory equipment, available via VLab to its users. In most cases such experiments will be interactive processes, with users manipulating directly the remote equipment via specialized control software GUI. Experiments are difficult to describe in a normative manner. Depending on a specific science domain, there can be many dependencies of external, often non-deterministic factors. The device will not be continuously available for VLab users but will be shared with local researchers (usually with higher priority than the remote users). There are also maintenance periods, in which the device is unavailable. Sometimes the presence and assistance of the device operator may be necessary — especially at the beginning of experiments.

The first scientific device incorporated into the VLab system was an NMR spectrometer. The most important problems with scheduling the NMR experiments, apart from those described above, come from the samples management. To perform an NMR experiment, an actual sample containing chemical compound has to be delivered to the NMR spectrometer and inserted into the machine. This causes a number of scheduling problems. At the task (and corresponding sample) submission point the actual NMR device has to be known and chosen, because the sample has to be sent from the remote location to the device site. The exact time of sample arrival is not known as well, making the exact task scheduling impossible until the sample arrives.

6.3.2. Digital Science Library for NMR

Current Digital Science Library (DSL) implementation has been used in the Virtual Laboratory for NMR spectroscopy purposes. Its main functionality, which is storing and presenting data in a grid environment, was extended with the functions specific to the requirements of the NMR data. This data can be the input information used for scientific experiments as well as the results of performed experiments. Another important aspect is the capability of storing various types of publications and documents related to NMR discipline, which are often created in the scientific process. This type of functionality, which is well known to all digital library users, is also provided by the DSL.

The analysis of one or multi-dimensional homo- and hetero-nuclear spectra obtained in the course of NMR experiment can provide information about the chemical shifts of the nuclei, scalar coupling constants, residual dipolar coupling constants and the relaxation times T1, T2. All of these data can be stored in the presented database, which also offers tools for performing quick and optimal search through the repository. Compared to the other NMR databases available through the Internet, like BioMagResBank, NMR data-sets bank, NMRShiftDB, SDBS and Spectra Online, DSL is more suitable for teaching, since it contains an entire range of the information about the perfomance and analysis of the NMR experiment.

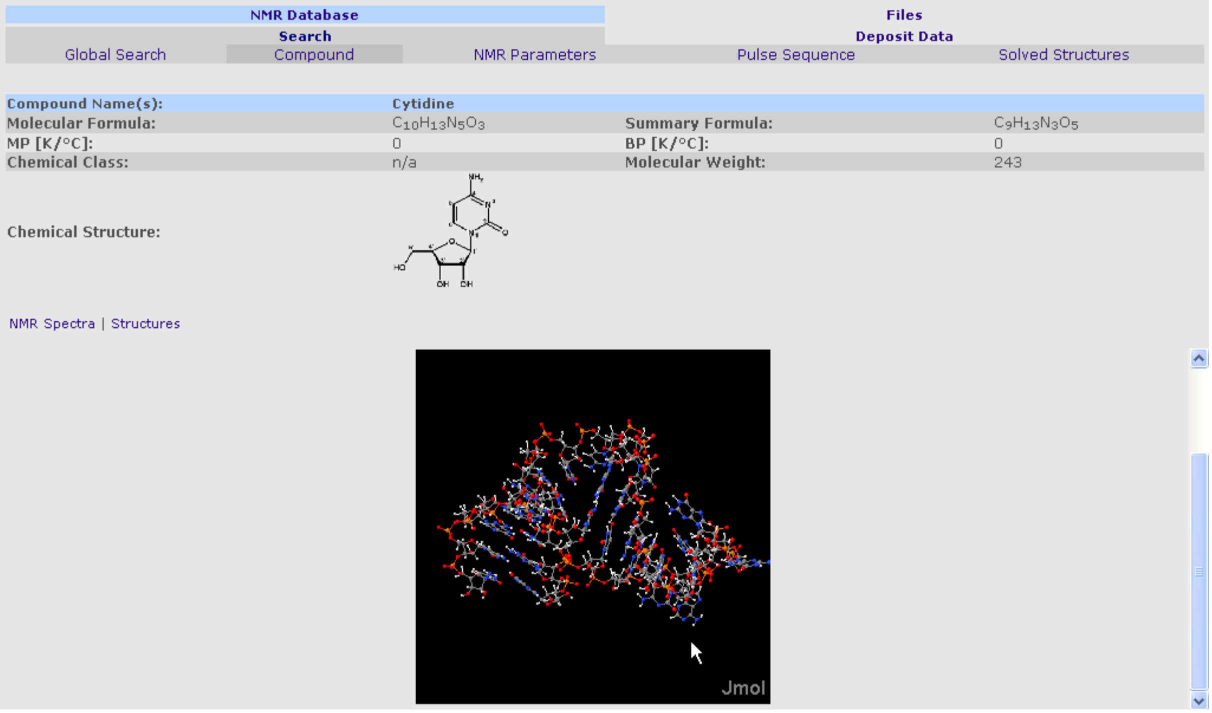
Information about each compound which is stored in the NMR DSL comprise its unique name, a chemical formula, information about its physiochemical properties and a specific description of all its atoms consistent with a HOSE code. Graphical representation of one- and multi-dimensional NMR spectra as well as their source files obtained from NMR experiments and a complete description of parameters used to acquire spectra are also kept in the repository. Additionally, it is possible to place the pulse sequence programs which can be used by all the database users. In case of biomolecules — proteins and nucleic acids — the database yields a possibility of depositing atom coordinates, structural constraints used in the computation and the complete structure determination protocols. References to all the stored data in a form of a list of papers are also included. A complete list of record fields and their formats is presented in the table below. Information is processed with the use of the following operations: enter, modify, search.

No	Name	Type	Size	Description
	Information about the compound			
1	Compound Name(s)	Text	256 characters	Chemical name(s) of the compound
2	Molecular Formula	Text	32 characters	Molecular formula
3	Summary Formula	Text	32 characters	Summary formula
4	Chemical Structure	Graphics		Molecular formula with atom numbering
5	Atoms name with numbers and HOSE code	Text	Array: N·16 (names, numbers of atoms) Array: N·64 (HOSE codes)	Atom names and their corresponding HOSE codes
6	Chemical Class	Text	256 characters	Chemical class of the compound
7	Molecular Weight	Text	16 characters	Molecular weight
8	Boiling Point	Text	16 characters	Boiling point
9	Melting Point	Text	16 characters	Melting point
10	Form Type	Text	2 characters	Type of the form
	Information about NMR experiment			
11	1D NMR Spectra	Graphics	~ 0,5 MB	Graphical representation of 1D NMR spectra
12	Correlation type	Text	16 characters	Type of correlation of 2D and 3D NMR spectra: homonuclear, heteronuclear, other
13	2D NMR Spectra	Graphics	~ 0,5 MB	Graphical representation of 2D NMR spectra
14	3D NMR Spectra	Graphics	~ 0,5 MB	Graphical representation of 3D NMR spectra
15	Pulse sequence	Binary	~ 0,25 MB	Pulse sequence
16	Pulse sequence — description	Text	64 characters	Description of the pulse sequence
17	Spectrometer model	Text	32 characters	Model of NMR spectrometer used in an experiment
18	Spectrometer manufacturer	Text	16 characters	Manufacturer of NMR spectrometer used in an experiment
19	Spectrometer basic frequency	Text	16 characters	Spectrometer basic frequency
20	Temperature	Text	16 characters	Temperature of experiment
21	Solvent	Text	32 characters	Solvent name
22	Concentration	Text	16 characters	Concentration of the compound
23	PH	Text	8 characters	pH of the solution
24	Buffer	Text	128 characters	Buffer contents and its name
25	Reference	Text	32 characters	Reference compound name and its chemical shift
26	Reference Compound	Text	32 characters	Example of reference compounds with their chemical shifts
27	Additional Spectrum Information	Text	Text file	Additional spectrum information
	Information about spectral data			
28	^{1}H, ^{13}C, … Chemical Shifts	Text	Array: N·8	Available chemical shifts
29	Coupling Constants — Scalar	Text	Array: N·64	An array of available values of scalar coupling constants
30	Coupling Constants — Residual Dipolar Couplings (RDC)	Text	Array: N·64	An array of available values of residual dipolar coupling constants
31	Relaxation Times	Text	Array: N·2·16	Values of relaxation times T1 and T2
	Information about the author			
32	Authors	Text	256 characters	List of authors
33	Institution	Text	128 characters	Institution
34	Contact	Text	128 characters	Contact to the author (e.g. e-mail, phone)
	Bibliography			
35	Authors	Text	256 characters	List of authors
36	Journal	Text	128 characters	Journal name
37	Year	Text	8 characters	Issue year
38	Volume	Text	16 characters	Volume
39	Pages	Text	16 characters	First and last page of the article
40	Title	Text	256 characters	Title of the article
41	PDF, PS	Binary		Paper in PDF or PS file format
42	Link	Text	128 characters	Link to the bibliographic base

	Experimental data			
43	Original NMR Data	Binary	~ 10 MB	Compressed NMR data
44	Format of data	Text	64 characters	Format of NMR data
45	Processed data (zipped)	Binary	~ 50 MB	Compressed processed NMR data
46	Format of processed data	Text	16 characters	Format of processed NMR data
	Structural analysis			
47	Protocol (zipped)	Binary	~ 1 MB	Protocols, scripts, ...
48	Protocol description	Text	512 characters	Description of a protocol
49	Restraints (zipped)	Binary	~ 0,2 MB	Experimental restraints
50	Restraints description	Text	256 characters	Description of experimental restraints
51	Structures (zipped)	Binary	~ 5 MB	Atom coordinates
52	Database	Text		Name of the database storing atom coordinates
53	Reference to database	Text	128 characters	Reference to database
54	Code	Text	16 characters	Structure code
55	Additional Data	Text	512 characters	Additional information
	Other information			
56	Other information	Text	512 characters	Additional information

Table 6.3.1: List of attributes present in NMR database.

Based on the data that are stored in the database such as chemical shifts, coupling constants or relaxation times, the user can quickly identify a molecule. In case of unknown compounds, the knowledge of chemical shifts and coupling constants together with a set of the HOSE codes can be used to predict the chemical environment of atoms and to suggest the types of functional groups which are present in the analysed molecule. The HOSE codes can also be helpful in the simulations of the spectra performed on the basis of molecular formula of the compound.



Compound Name(s):	Cytidine		
Molecular Formula:	$C_{10}H_{13}N_5O_3$	Summary Formula:	$C_9H_{13}N_3O_5$
MP [K/°C]:	0	BP [K/°C]:	0
Chemical Class:	n/a	Molecular Weight:	243
Chemical Structure:			

Figure 6.3.2: NMR digital library in Virtual Laboratory.

6.3.3. The Use-Case Diagram for the DSL-NMR Library

We can distinguish the main actor on the base of demands defined for the Digital Science Library of Nuclear Magnetic Resonance Spectroscopy. It is presented in figure 6.3.3. As an actor we can consider the system placed in the presentation or the indirect (middle) layer, on the condition that it is able to communicate over the SOAP protocol.

The following diagram presents two actors. The first — SOAPCompliedClient — is identified with a certain pattern of actor with granted rights for executing the specified operations and provided with the ability of intercommunication with the library over the SOAP protocol. The nuSOAP (nuSOAP_PHPClient) is the second actor in the diagram. It can be used to create an access service for the NMR Library. In this case the access service is based onWWW pages and the PHP programming language.

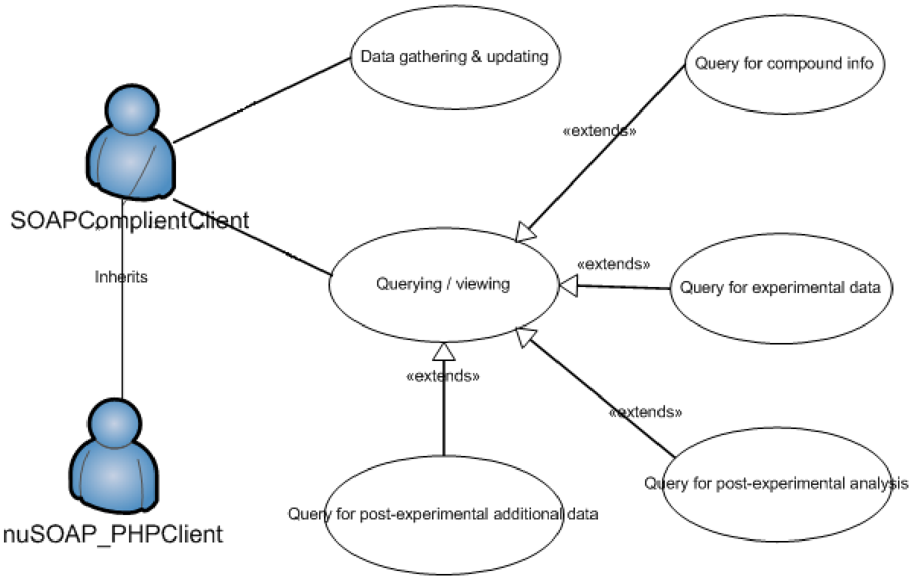

Data gathering & updating
Query for compound info
«extends»
SOAPComplientClient
Inherits
Querying / viewing
«extends»
Query for experimental data
«extends»
«extends»
nuSOAP_PHPClient
Query for post-experimental analysis
Query for post-experimental additional data

Figure 6.3.3: Use case diagram for DSL-NMR.

The diagram shows some main examples of using the DSL-NMR library system:

- **Data gathering and updating:** enables introducing and updating information related to the chemical compound and theexperiment data,
- **Querying and viewing:** enables getting information on a chemical compound, experiment and its results, and the analysis executed on the basis of received data.

The search method depends on the datarequired. Typical queries are:

- **Compound info.** Prerequisite: Existence of information concerning re-searched chemical compound; searching out its attribute;
- **Experiment data.** Prerequisite: Existence of experiment data; searching out after attributes of an experiment, for instance the spectrum type, sequence of impulses, spectrometer type, solvent, measurement temperature;
- **Post-experimental analysis.** Prerequisite: Existence of the analysis description; searching out of chemical compound according to value of chemical shift, coupling constant;
- **Post-experimental additional data.** Prerequisite: Existence of the re-search documenting data, e.g. reference for authors of experiments, for articles concerning led research or papers documenting the received results.

6.3.4. References

Rek, P., Kopec, M., Gdaniec, Z., Popenda, L., Adamiak, R.W., Wolski, M., Lawenda, M., Meyer, N., Stroinski, M.: Digital Science Library for Nuclear Magnetic Resonance Spectroscopy. The 4th Cracow Grid Workshop, ISBN 83-915141-4-5, pp. 404-411, Cracow, Poland, December 12-15, 2004

7. Summary

This deliverable presents state of the art of grid middleware and corresponding emerging standards for sharing scientific instruments over international networks. It does this by describing the existing middleware in depth (chapter 3) and by providing in-depth knowledge of related standards, both existing and forthcoming ones (chapter 2). Of particular interest in this regard is the CIMA project, a US-founded project which has similar aims as the European GridCC project.

Some of the middleware is already deployed in several testbeds. We aim to describe the two most important ones in chapter 4: We concentrate on the Virtual Laboratory project as well as the GridCC project. The Virtual Laboratory shows how a workflow can look like when conducting an experiment. The GridCC project is particularly interesting because of its "Instrument Element", a middleware component which represents an instrument in the grid infrastructure. It is expected that some components from these projects will be used when deploying a remote instrumentation infrastructure within the European Union.

What kind of middleware is suitable for deployment will be evaluated in chapters 5 and 6. These chapters present exemplary use cases where one can see what requirements are fulfilled by current technology and what is still missing. While this workpackage concentrates on what is currently available, WP4 will elaborate on the missing parts.

While this deliverable does not present a conclusive table where all requirements are contained, there will be such a table when the prototyping phase (WP6) starts: Deliverable 3.3 will present such a summary.

References

The references are contained at the end of each individual section for better text comprehension.

Contact Information

All authors affiliation:
Poznań Supercomputing and Networking Center
ul. Noskowskiego 10
61-704 Poznań, Poland

URL: http://www.man.poznan.pl
Tel. (+48 61) 858-20-00
Fax (+48 61) 852-59-54

Davide Adami	davide.adami @ cnit.it
Antonio-Blasco Bonito	blasco.bonito @ isti.cnr.it
Marius Branzila	branzila @ ee.tuiasi.ro
Luca Caviglione	luca.caviglione @ cnit.it
Romeo Ciobanu	rciobanu @ ee.tuiasi.ro
Davide Dardari	davide.dardari @ deis.unibo.it
Franco Davoli	franco.davoli @ cnit.it
Marcio Faerman	marcio @ rnp.br
Alberto Gotta	alberto.gotta @ isti.cnr.it
Damian Kaliszan	damian @ man.poznan.pl
Constantinos (Costas) Kotsokalis	ckotso @ admin.grnet.gr
Angelos Lenis	anglen @ netmode.ece.ntua.gr
Fabio Marchesi	fabio-marchesi @ libero.it
Thomas Prokosch	tprokos @ gup.uni-linz.ac.at
Tomasz Rajtar	ritter @ man.poznan.pl
Michael Stanton	michael @ rnp.br
Zhili Sun	z.sun @ surrey.ac.uk
Jin Wu	jin.wu @ surrey.ac.uk
Tasos Zafeiropoulos	tzafeir @ grnet.gr
Yanbo Zhou	y.zhou @ surrey.ac.uk