Chapter 5

Information Management for Scientific Applications

5.1 Introduction

This chapter identifies the basic information management requirements of emerging applications in e-science and presents the main approaches taken within the Virtual Laboratory project addressing these requirements. The Virtual Laboratory project (VL\textsuperscript{1}) initiated at the University of Amsterdam and supported by the Dutch ICES/KIS-II program, aims at the design and development of a generic environment and a flexible architecture supporting scientific applications and scientists with their experiment definition and control, data handling facilities, and access to distributed resources. The cooperative federated information management framework developed by the CO-IM group for the VL, aims at providing the necessary information services to enable scientists and engineers to work on their experimentations, and to properly handle all related data/information [AKB\textsuperscript{+}01, ABK\textsuperscript{+}00]. However, the work presented in this chapter describes only the contribution of this author to the partial design and development of the VL information management functionalities.

This chapter first briefly describes the architecture design of the Virtual Laboratory and then focuses on specific advanced features, functionalities, and facilities introduced and developed for management of information in scientific applications. The addressed features include:

- The **strategies** for storage and retrieval of multimedia scientific information (addressed in section 5.3.1),
- The **use of standards** for scientific data modeling and archiving, supporting the integration of data from heterogeneous sources (addressed in section 5.3.2).
- **Universal and schema free access** to scientific data, stored within various local and remote database management systems (addressed in section 5.4).
- **Access security** to the data available within the VL archive is based on **predefined visibility restricted schemas**. As such, appropriate access rights and visibility levels

\textsuperscript{1}Virtual Laboratory (VL) is also referred to as VLAM-G that stands for Grid-based Virtual Laboratory Amsterdam.
for individual users and groups are presented. This approach, addressed in section 5.5, is mainly adapted for *VL Scientific Results Publishing.*

At the end of this chapter performance issues of the suggested implementation approach is addressed. *Benchmarking tests,* for the storage/retrieval of massive amount of data is performed. The presented results address the specific features that assure database efficiency and performance, including the information access security and the short response time for data transfer (addressed in section 5.6).

If we define a site as an organization with several application systems and databases, *this chapter provides generic tools and advanced facilities that can be adopted by every site to enable it as a node in a cooperation network.* On one hand, sites can benefit from these generic tools and advanced facilities in order to make their applications and databases stronger and more efficient. On the other hand, the use of standards and the consideration of state-of-the-art techniques, when developing these tools, enable these sites for appropriate collaborations taking advantages of (1) interoperability for data access and communication, (2) information sharing/exchange for interfacing and federation, and (3) collaboration within Virtual Laboratories and Virtual Organisations.

Later on in Chapter 6, the design and development of a more generic and flexible collaborative information management framework is described, addressing the cooperation among different centers and scientists, required within the e-science and other emerging collaborative applications.

### 5.2 Virtual Laboratory Architecture Design

A main goal of VL is to provide a science portal\(^2\) for distributed data analysis in applied scientific research. The VL supports scientists with all the steps involved in conducting their experiments, using the Virtual Laboratory facilities. The steps may involve experiment definition and control, access to local and remote sites, process and analyze the retrieved data, archive and publish the resulted information, control of external devices from their experiments, use advanced tools to simulate and visualize the results, collaborate with other scientists and centers, and so on. Thus, Virtual Laboratory (VL) provides a *user-oriented environment and a science portal, supporting collaborative, Grid-based distributed analysis in applied sciences, using cross-institutional integration of heterogeneous information and resources [ABB+01].*

The general design of the VL architecture is based on multiple functionality layers, so that the required application-specific and domain-specific computational and engineering features can be separated and dealt with differently from the generic computing and data management aspects. The generic aspects serve a broad range of scientific domains, and include features related to parallel/distributed computing and networking infrastructure, basic middleware tools for information management and collaboration, and generic environment for simulation and visualization techniques. Further, domain-specific or application-specific features, are defined on top of these generic features.

---

\(^2\) A science portal refers to delivering science information and services to industry, investors, and to the research community; using cross-institutional integration of heterogeneous information and resources.
As illustrated in Figure 5.1, the VL reference architecture is primarily composed of four functional layers [ABK+00, AKB+01, ABB+01]. This multi-layered design primarily represents the functionalities supported by the VL, and does not imply that the development of the layers at the “upper” levels is dependent on the “lower” layers. Rather, it represents the fact that on one hand every layer can be developed simultaneously and independently of the others without the need for extensive interaction during the design. On the other hand, it allows to focus on one layer at a time and it provides possibilities for a clear description of the primitive Virtual Laboratory operations and components, and their individual functionalities at different levels. The four-layer architecture of VL is briefly described below.

Layer 1: Grid Resources Layer

The Grid Resources Layer provides the high-bandwidth low-latency communication platform, which is necessary both for accessing the underlying large data sets and for the physical or logical distribution of the connected external devices and the client community that uses the laboratory facilities.

Layer 2: Grid Services Layer - Grid Middleware

The gigabit networking technology being set at the University of Amsterdam, and the Globus distributed resource management system [FK 98], are used for the development of VL Grid middleware environment. The Globus system addresses the needs of the high-performance applications that require the ability to exploit diverse, geographically distributed resources. The VL Grid Middleware provides basic mechanisms for the communication, authentication, network information, and data access. These mechanisms are used in VL to construct various higher level services, such as parallel programming tools and schedulers for multidisciplinary scientific applications. Since the Globus system offers the resource management required for distributed computing, it is used for the development of some
Layer 3: Application Toolkit Layer - VL on Grid MiddleWare

Two main roles are targeted by the development of the VL middleware layer. First, it provides a set of generic functionalities for the various applications from advanced scientific domains such as physics, chemistry, system engineering, medicine, and biology. Second, it bridges the gap between those advanced applications and the Grid-services layer. The major components of VL middleware support unique information provision and provide means for resource integration and collaboration. These components include:

1. The VL] Portal and RTS. provides a Grid-based science portal, supporting the VL functionalities for a wide range of users, while hiding the details related to the distributed data computation from the end-users [BHG+01]. The VL portal can be easily used to define a new VL application, while the RTS (Run Time System) forms the VL core component constituting the interface between the VL Grid MiddleWare and the underlying Grid services (Grid MiddleWare).

2. The VL Information Management for Cooperation (VIMCO Module) provides archiving services as well as the information handling and data manipulation within the Virtual Laboratory. This module supports a wide range of functionalities ranging from the basic storage and retrieval of information (e.g., for both the raw data and processed results), to advanced requirements for intelligent information integration and federated database facilities, and to supporting the collaboration and information sharing among remote centers.

3. The Communication and Collaboration (ComCol Module) enables the communication with external devices connected to the laboratory, as well as the secure communication and collaboration between users within and outside the laboratory.

4. The Virtual Simulation and Exploration (ViSE Module) presents a generic environment in which scientific visualization, interactive calculation, geometric probing and context-sensitive simulations are supported.

Layer 4: Application Layer

At the top layer of the architecture is the application dependent part of the Virtual Laboratory framework. Within this layer, interfaces are present, and application-specific and domain-specific tools are provided in order to enable users to make their specific experiments, using the functionality provided by the other layers in the architecture. Among the domain specific application cases that are currently in development phase within the VL project, we enumerate: MACS- material analysis for complex surfaces [FAE+01, EAG+01], EXPRESSIVE- genome expression in biology [KAB+01], dynamic exploration and distributed simulation within interactive environments [BS00, SKH+99], and EFC - Electronic Fee Collection and intelligent transport systems [VWH00, DHA+98, HDB+97].

Due to the focus of this chapter of the thesis on the information management within VL, the following sections only describes the contribution of this author to the partial design and development of VIMCO module.
5.2.1 The VL Information Management for COoperation - VIMCO Module

The VIMCO module is being designed as a multi-level information management system and environment to support the classification and manipulation of the data within the Virtual Laboratory environment. Considering the wide variety and large amount of data handled within different layers of the VL, the required information management mechanisms may vary. Namely, the need for parallel database extensions, distributed database facilities, and federated/integrated information management; must be considered. These extensions are necessary to better support the information management requirements of advanced scientific applications. Therefore, the design of the information management system must support structured as well as binary data access, data integration from several sources, location transparency for remote data access, secure and authorized access to shared data among networked applications, and the intelligent data handling.

The general design objectives of the VIMCO system within the VL cover the areas of fundamental database research and development to support complex domains. The VIMCO development primarily addresses two main focus areas:

1. The first area focuses on the **Data Archive**: storage and retrieval of a wide variety of scientific and engineering data necessary to be handled within the VL, supporting their categorization, storage, and scientific data publishing (that is the focus of this chapter).

2. The second area concentrates on the development of a **Generic and Flexible Information Integration System (GFI$_2$S)**: a flexible collaborative framework preserving systems autonomy and supporting the import/export of data based on information visibility and access rights defined among systems (that is the focus of Chapter 6). GFI$_2$S is designed as a generic approach that serve the information integration in a highly dynamic network of applications. Therefore, its deployment within the Virtual Laboratory environment can improve the accessibility to large databases for data intensive applications and can provide access to a variety of distributed sources of information.

The work on data archiving focuses on the design and development of an information brokerage system to archive the wide variety of data with different modalities and from different sources. This includes all the data generated through specific research and application domains supported by the VL framework. For instance, for the information handled by ViSE and ComCol modules of the VL as well as other VL applications, a catalogue/archive schema has been developed using the Dublin Core MetaData standard. This catalogue/archive schema has been refined to achieve a more scalable and extendable archive meta-metadata, able to capture comprehensive information about the complete experimentation process (e.g., raw/processed data, experiment parameters, scientist information, hardware devices, and software characteristics). The designed schema is extendable to cope with the future modifications and with the flexible addition of new experiment types.

5.3 Multi-Media Scientific Data Sets Manipulation

One common characteristic of the scientific domains such as biotechnology, physics, astronomy, and complex engineering applications, is that they all produce large data sets. The
data generated from different experiments in each of these domains needs to be inter-linked and referenced, so that the scientific applications can fully utilize the outcomes of the experiments.

The physical data storage approach plays an important role in the long-term strategy for data management in organizations. The data storage approach chosen for an application environment mostly depends on the requirements of the application, specifically for data archiving and information access. Simple applications, for instance, are built on top of the file system. Other applications from system engineering domain, however, require the deployment of proper physical database design for storing their data. Nowadays, more complex applications such as in bio-informatics [Gelb 98], biology [BDH+95], and medicine [BAS+99] require much more advanced solutions. In such solutions, the design of a proper system architecture and physical database approach can help in solving problems related to information security and efficiency of access, as well as facilitating the cooperative working processes among different experimenters and sites. The approach must also take benefits from using database systems in terms of user view definition, information sharing, and system integration.

Considering the complexity and the distribution of data in advanced scientific applications, the proper management of the domain information and knowledge is challenging. The scientific data storage/archiving and data access/retrieval mechanisms must be addressed in such a way that data sets can be properly searched, retrieved, compared to other existing data sets, published, and inter-linked. In addition, the necessary mechanisms for information security, performance issues, and the means to distinguish and protect the private data, together with the necessary support for the data to be published and shared with remote users must be provided.

Therefore, the objectives of a system for scientific data management must go beyond just providing a networked "hierarchical storage management" system [HSM 01], which only enhance the notion of a traditional file system made up of a hierarchy of directories and files. Traditional mechanisms for storing large volumes of scientific data are inadequate to satisfy the long-term cataloging, access and retrieval needs of scientific experimentations and their meta-data.

In scientific applications, meta-data refers to the annotations and added "information" to the scientific data sets, which is for instance different than the meaning of meta-data, referring to schemas in databases area. Scientific meta-data is an essential component of a data archive; its storage and maintenance help users to understand the structure of the archive and provide necessary information to correctly interpret the data [JCF 95]. Scientific applications commonly include a metadata database utilizing database management systems to provide users with a powerful query facility. The metadata database is often managed separately from the archive as a facility for locating data [JCF 95, GSB 95]. In many cases, recent research combines DBMS technology with the so-called hierarchical storage management systems (HSM). The DBMS is re-engineered to use an IHS as its storage medium resulting in metadata databases with virtually huge data sets located at various storage facilities [SS 95, BFL+95].

The Scientific Data Management approach in [HA 96, AHW+98], built on top of the traditional file system, addresses this challenge by providing a hierarchical storage management system to store data files, as well as a database that captures information about those files. Thus, it provides researchers with enhanced facilities for storing, locating, retrieving, and interpreting archived data. Similarly, the Intelligent Archive (IA3) provides scientists

---

3Intelligent Archive, Lawrence Livermore National Laboratory - LLNL (http://www.llnl.gov/ia/).
with advanced capabilities for organizing and searching the information. One drawback to these approaches is that keeping the metadata database synchronized with the archive can be difficult.

5.3.1 Storage of Large Scientific and Engineering Data Sets

So far, depending on the requirements and criteria, a variety of approaches are applied to the storage of such data. We categorize the approaches discussed in this section, addressing the management of large-scientific multi-media data sets, into:

1. The traditional file system approach, described in section 5.3.1.1.

2. The external data link approach, defined for access to external data sources, described in section 5.3.1.2.

3. The one-database storage approach, described in section 5.3.1.3.

The file system approach, traditionally used in the past, has shown its inefficiency especially in maintaining the links between the inter-related information pieces, searching the available information, and comparing results of different experiments. For instance, consider a group of scientists that every day perform different, but inter-related experiments within the bio-informatics\(^4\) application domain. Not only the generated experiment results are complex and cannot be managed using a relatively simple file system, but also the complete set of information concerning each experiment cannot be properly inter-linked in order to give the entire experiment more value. Furthermore, the results of an experiment cannot be fully exploited if information about the experimenter is missing, neither if the input data sets or environment parameters are not coherently available.

The second technique, external data link approach is much more effective than flat files for storing and managing large data sets. In this case, a database catalogue is used together with the file system, to efficiently manage distributed scientific information. The database will provide references to all objects that are stored locally or remotely at geographically distributed sites. This approach solves the problems related to the database overloads with huge objects and improves the system performance. The advantages of this approach includes, among others, the provision of a mean for managing huge amounts of data, provision of data in a secure manner, and distribution of data among geographically distributed nodes: where the nodes are usually those in which the data is generated or where it belongs to. [BAH 99, PWD\(^+\) 99].

The last approach, the so-called one-database storage approach, consists of storing both the binary data and the other general information of the application, into the same database. However, talking about binary data means data of very large size, which results degradation of performance when loading this huge data into the database itself. The database catalogue needs to be much better exploited for metadata cataloging, indexing, and proper searching facilities.

However, the emerging scientific and large engineering applications generally require more than what is provided by these approaches. New approaches need to be developed for the management of large data sets in order to improve the performance of the system while preserving the consistency of data and information visibility levels. One possible solution, suggested in this section, is to merge and extend the second and the third approaches

\(^4\)Bio-Informatics is an interdisciplinary science that studies and explores biological issues and cases using and benefiting from the methods of informatics.
presented above into one distributed system consisting of a front-end catalogue database and a back-end distributed database server. In this approach, described in section 5.3.1.4, and referred to as the “parallel/distributed database server approach”, the meta-data (description of the data) and the annotations are kept in the database catalog, while the large multimedia data sets are stored at the distributed servers [BCG+97, WMP 98]. Queries on the data data are formulated against the database catalogue, and since this is generally a small amount of data, a good performance can be achieved by the query processing.

Following sections describe different strategies for data storage and mainly focus on an extended approach, which deploys a parallel/distributed database server for on-line object delivery to authenticated users/applications.

5.3.1.1 File System Approach

File systems store information in O/S files, and allow the storage of a very large amount of data over a long period of time. However, these files are in different formats and the programs accessing them are coded in different languages, which may result in data redundancy and inconsistency. Data in files is not automatically backed up, in order to guarantee its availability, a recovery system must be developed and set up [GUW 02].

The file system approach is still used by applications in which, the implementation is based on the manipulation of regular files. The developed application programs are fully dependent on the data files (see Figure 5.2). Thus, the structure of data files is embedded in the access programs and any changes in the structure of a file require re-compiling all programs that access this file. This approach, used by several applications in the past, cannot fully support many application requirements of the advanced scientific domains, where information needs to be inter-linked, compared to other data, and easily accessed. The file system by itself is not able to handle complex inter-linked huge data sets. Thus, applications based on file systems are hard to maintain and to extend [BAK+00, EN 00, GUW 02]. The inefficiencies of this approach are evident in:

- Maintaining the link between the inter-linked pieces of information, and comparing related data in different applications,
- Supporting the ability to query and modify the data using an appropriate query language, and their support for schema being limited to the creation of directory structures for files.
- Searching the stored information and supporting access to data items whose location in a particular file is not known.
- Preserving the system coherency, data scattered in various files of different formats.

Figure 5.2: File System Approach
Furthermore, the file system approach cannot properly support the requirements for:

- Security for access to data, which can only be enforced at the level of operating systems and not further.
- Critical concurrent access control situations to files by several users, which may not be prevented.
- Data "sharing" though the Internet, due to the lack of proper security for access to data and concurrency control mechanisms.
- Individual user views on the data are not supported, and data items, for which the location within the file is not known, are difficult to locate and retrieve.

However, for simple cases inside one organization, where the data structure and the application requirements are simple, well defined, and not expected to evolve, it may be more desirable to use regular files. The usage of a database management system in such a situation may involve unnecessary overhead costs that would not incur in traditional file processing [EN 00].

5.3.1.2 External Data Link Approach

The external data link approach uses a database catalogue together with the file system, to efficiently manage distributed multimedia data sets. In the external data link approach, a single repository (catalogue) for meta-data and general information is maintained. The database catalogue (also referred to as metadata database) is defined once and then accessed by various users and applications. Scientists can use the metadata to locate and interpret data stored in the archive, including data generated by other scientists. This metadata provides permanent documentation of the data and becomes an integral part of the scientific data management system. Thus, the database catalogue stores the structure of the data, documents the contents and context of the data stored in the archive, and references the large binary objects that are available either locally or remotely at geographically distributed sites (Figure 5.3). This approach solves the problems related to centralized database overloads with huge objects, provides better retrieval performance, and improves the access to the data through a single database catalogue.

![Figure 5.3: External Data Link Approach](image)

In addition to the main characteristics listed above, the usage of this approach provides scientific applications with the following advantages [BAH 99, PWD+99]:

• Data is stored separate from the access programs (program-data independence).
• Data is stored at one place, either at the point where it is generated or where it belongs.
• Data is distributed so that it can be physically located closest to its intensive usage.
• Data is visible from any node within the cooperation community.
• Data distribution reduces the access bottlenecks at individual sites.

However, there are of course several limitations associated to this approach. The first problem faced when using this approach relates to the database catalogue consistency. In general, the referenced external objects are stored independently of the database catalogue. This later only contains a simple link to the external objects that are stored as regular files on different local and remote systems. Thus, these files can be updated or removed by local users within each system, without notifying the database catalogue maintainer, which may result to inaccurate or incorrect reference links.

The problem of database catalogue consistency could be addressed by developing a specific module that automatically and periodically checks the availability of the referenced objects against the database catalogue content, then the database catalogue will be updated regarding the new changes. If an external referenced object is missing for instance, the link to it will be removed from the database catalogue in order to avoid inaccurate references. Furthermore, log information can also be gathered and stored. The log information can support keeping track of the system updates based on some comparison of the size, the date, and the author of the referenced external objects.

A second problem concerns the security issues for the stored objects, where a more critical situation relates to data privacy and user visibility rights. Under normal considerations the files are usually hold in a public location, so when users request information from the system, their requests are evaluated against the database catalogue and they will be provided with the links to the proper objects that they can access and retrieve via http, ftp, and other data transfer protocols. Thus, typically the objects are not secure.

To solve the security problem within this approach, some applications develop a number of remote file servers, through which, the file access and user authentication are controlled based on the database catalogue information [BAH 99, PWD 99].

5.3.1.3 One-Database Storage Approach

As depicted in Figure 5.4, the one-database storage approach consists of storing the binary objects together with their meta-data and the other general information of the application, within the same database (being centralized or distributed) in a unified way, and typically in digitized format. This approach requires substantial re-engineering or extension of traditional DBMSs to directly query and manipulate the contents of the files stored in the database. However, this problem is not a barrier for research anymore, since several database systems (e.g. Matisse5, Oracle6, Jasmine7, Informix8, and DB29) already support the storage of multi-media and binary objects of different formats (e.g. postscript, images, audio, video, documents, etc.). The storage of these types of data is made possible by the DBMSs via a set of binary large objects (the so-called blobs).

5Matisse: The Object Developer's Database System (http://www.fresher.com).
6Oracle Database System (http://www.oracle.com).
7Computer Associates'-Jasmine (http://www.cai.com)
8Informix® information management solutions (http://www.informix.com)
9IBM DB2 Universal Database System (http://databases.about.com/cs/db2/).
The one-database-storage approach solves the problem of keeping the metadata database synchronized with the archive, by making the metadata and the archive one-and-the-same.

However, here the binary data refers to data of huge size, where even the loading of these objects in the database affects the database performance, which can be better exploited for cataloging, indexing, and searching facilities. Another problem may be the access mechanisms to these objects, which require extra encoding/decoding facilities (embedded plug-ins) to be supported by the database software in order to properly manage the compact data formats of the binary objects.

In addition, the studies made on engineering and management of data from different applications in the domain of music industry [BAH 99a, BAH 99b], water distribution management [ABH 98a], biology and medicine [BAS+99], show that the information dealt with is of two main categories: the general information (so-called meta-data) and the raw data. The meta-data represents the description of a wide variety of data, which is accessed by a large number of users; where the user queries are issued against the meta-data. The raw data, on the other hand, is generally accessed and processed by the end-user scientists and experimenters.

From the usage point of view, the one-database storage approach that stores large binary objects in the same database together with the general information of the application is easier and more desirable. However, it reduces the database performance and efficiency. Let us consider one example application that is now being developed at the University of Amsterdam. This application is focused on the management of large data sets and the information handling in a bio-medical\footnote{Bio-medical Engineering is an interdisciplinary field between medicine and engineering, which applies the most advanced technologies, principles, and skills developed in engineering to the world of medicine.} application [BKS 98], where the data originates for instance either from a simulation experiment or from medical scanners (CAT, MRI)\footnote{CAT stands for Computerized Axial Tomographic and MRI stands for Magnetic Resonance Imaging.}. In such applications, the complete information about scans are “slice wise” stored in the database, and delivered at run-time, on demand. The largest objects that are stored in the database are the raw data sets for the slices, used for on-line simulation, visualization and exploration. Considering a normal bio-medical application that deals with dynamic grids of $1024 \times 1024$ 16-bit pixels with 5000 slices over 50 time steps, the amount of raw data reaches several hundreds of gigabytes (around 524 GB). Our detailed study on these applications also shows that the general information only requires small disk storage capacity compared with the raw data that is, in most cases, in the range of hundreds of times larger.

![Figure 5.4: One-Database Storage Approach](image)
5.3.1.4 A New Approach: Parallel/Distributed Database Server

In many collaborative application domains, for instance, the electronic commerce and the large scientific applications, access to databases is concentrated on large structured (and unstructured) objects of "value", and their related information (meta-data). In order to gain better database performance and access efficiency, learning from previous approaches, an alternative solution would be the utilization of a "distributed database server" to manage the large binary objects, separate from the general information. The proposed architecture deploys a parallel/distributed database server, that delivers objects (files) based on user authentication [BAH 99a, PH 98].

Figure 5.5 illustrates the architecture deployed within the parallel/distributed database server for scientific applications, where the followed strategy for data management takes advantages of both the external data link approach and the one-database storage solution. This approach uses a database repository to store the general information and a parallel/distributed database server (instead of file server) to store large objects. The utilization of database systems instead of file servers in such architecture enforces the issues related to security for access, concurrency control, and information visibility rights mentioned in previous sections 5.3.1.1 to 5.3.1.3.

![Database System Diagram](image)

**Figure 5.5: Architecture for the Parallel/Distributed Database Server**

The parallel/distributed database server is well suited when high security is required (objects are encrypted in the database as lists of bytes that are difficult to access due to database security and hard to decode since they require special encoders/decoders that are only delivered to authorized users). This feature is very important for scientific applications, since it assures a minimum level of security for the private data, where users of the system will also be authenticated by the database server before getting served.

The parallel/distributed database server provides the proper base framework that can be adapted to handle huge amounts of raw scientific data. With this architecture, the database nodes of the distributed server are inter-connected, making it possible for specific experimenters to connect to any database server within the distributed server, and request an object, without the need to know where the object actually resides. If the request cannot be handled locally at that database node, it is automatically broadcasted to the other database nodes, and the result is forwarded back to the user.
The followed approach is also suitable in the sense that the database can be used for different purposes. Figure 5.6 presents an application case of this architecture, in which, for instance in a Virtual Laboratory environment, the outside users and application interfaces are based on, and supported, through the database catalogue, while the scientists’ experimental interfaces are based on and supported by both the database catalogue and the parallel/distributed database server.

5.3.2 Scientific Data Archiving and Cataloguing Using Dublin Core Standard

Scientific and industrial organizations are nowadays focusing on building technology infrastructures that are cost-effective and conform to their application practices, in which the scientific meta-data plays an increasingly important role, and brings a considerable value in terms of information retrieval, application maintenance, data integration, and support for user requests. Such infrastructures maximize the sharing and re-use of data, eliminate redundancy, and facilitate application integrity [DCMI 99, BA 00].

To improve the applications’ flexibility and the end user’s usability, current systems must provide information about the content and the quality of the data they hold and manage. This information can be provided through the so called meta-data repositories [BA 00]. Scientific meta-data is information about the application data, that gives descriptive information about the context, conditions, and characteristics of the data. Thus, meta-data serves as the binding mean that ties the various tools and technologies together at the application level.

This section aims at the design and development of a database system, to archive a wide variety of large scientific data sets. The process of data storage/acquisition, from different sources, within scientific applications, can follow one of the two approaches, which are presented and described in section 4.5.1.

1. The first approach builds a specific two-side-dependent interface, to directly store the data from its origins into the corresponding databases.
2. The second approach uses an intermediate step by storing the data in an intermediate standard format (probably OIF and/or XML). The standard format can be loaded into the compliant database systems.

The main focus of this section is to apply the second approach to scientific data, using the Dublin Core Meta Data Standard [WKL+98]. The Dublin Core (DC) standard describes better the content of scientific data and the context related to its generation. This section addresses the design of a data archive, using Dublin Core, as follows:

1. First it gives a brief description of the Dublin Core Meta Data Standard: a specific higher-level cataloguing/archiving schema for the scientific raw/processed data.
2. Second, it addresses the object-oriented representation of the Dublin Core meta-data: a representation that better suites to scientific applications and makes the definition of the DC elements and the relationship among them more elaborate.
3. Third, it extends the object-oriented DC model with additional meta-data terms; in order to support the scientific applications regarding their experimentation and data processing.

### 5.3.2.1 Dublin Core based Meta Data Design and Implementation

The DC Meta-data Initiative [WKL+98] is a cross-disciplinary international effort to develop mechanisms for the discovery-oriented description of diverse resources in electronic environment. The Dublin Core Element Set comprises fifteen elements [DCMI 99], which together capture a representation of essential aspects related to the description of data sources (e.g. publishing). The meta-data definitions, presented in [DCMI 99], and provided in Table 5.1, include both the conceptual and representational form of the Dublin Core elements. The **Label** provides a mnemonic single-word specification for the DC meta-data **Elements**: labels are simple enough to identify the corresponding elements in the schema. While, the **Definition** captures the semantic descriptions for these elements. For simplicity reasons, some detailed definitions are summarized.
<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Label</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Title</td>
<td>A name given to the resource, by which the resource is formally known.</td>
</tr>
<tr>
<td>Creator</td>
<td>Creator</td>
<td>An entity primarily responsible for making the content of the resource. The Creator entity can be a person, an organization, or a service.</td>
</tr>
<tr>
<td>Subject &amp; Keywords</td>
<td>Subject</td>
<td>The topic of the resource content, expressed as keywords, key phrases or classification codes.</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td>An account of the resource content. Description may include but is not limited to: an abstract, a table of contents, reference to a graphical representation of content or a free-text account of the content.</td>
</tr>
<tr>
<td>Publisher</td>
<td>Publisher</td>
<td>An entity responsible for making the resource available. Examples of a Publisher include a person, an organization, or a service.</td>
</tr>
<tr>
<td>Contributor</td>
<td>Contributor</td>
<td>An entity responsible for making contributions to the content of the resource. Examples of a Contributor include a person, an organization, or a service.</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
<td>A date associated with an event in the life cycle of the resource. Typically, the Date is associated with the creation or availability of the resource. The date value is defined in a profile of ISO 8601.</td>
</tr>
<tr>
<td>Resource Type</td>
<td>Type</td>
<td>The nature or genre of the resource content. Type includes terms describing general categories, functions, genres, or aggregation levels for content.</td>
</tr>
<tr>
<td>Format</td>
<td>Format</td>
<td>Specifies the physical or digital manifestation of the resource. Format may include the media-type or dimensions of the resource and may be also used to determine the software, hardware or other equipment needed to display or manage the resource.</td>
</tr>
<tr>
<td>Resource Identifier</td>
<td>Identifier</td>
<td>An unambiguous reference to the resource within a given context. Example of formal identification systems include URI, URL, ISBN, and DOI (Digital Object Identifier).</td>
</tr>
<tr>
<td>Source</td>
<td>Source</td>
<td>A Reference to a resource from which the present resource is derived. The resource is referenced by means of a string or number conforming to a formal identification system.</td>
</tr>
<tr>
<td>Language</td>
<td>Language</td>
<td>A language of the intellectual content of the resource. The values of the Language element include a two-letter Language Code (ISO639), followed optionally, by a two-letter Country Code (ISO3166). For example, 'en' for English, 'fr' for French, or 'en-uk' for English used in the United Kingdom.</td>
</tr>
<tr>
<td>Relation</td>
<td>Relation</td>
<td>A reference to a related resource. The resource is referenced by means of a string or number conforming to a formal identification system.</td>
</tr>
<tr>
<td>Coverage</td>
<td>Coverage</td>
<td>The extent or scope of the content of the resource. Coverage includes spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity).</td>
</tr>
<tr>
<td>Rights Management</td>
<td>Rights</td>
<td>Information about rights held in and over the resource. The Rights element defines the rights management statement for the resource, or references a service providing such information. Rights information often encompasses IPRs and Copyrights.</td>
</tr>
</tbody>
</table>

Table 5.1: Dublin Core: Elements Description
5.3.2.2 Dublin Core Object Modeling

Current implementations for the Dublin Core model are addressed by "pseudo-hierarchical dot notation" (e.g. DC.Creator.Email) or in best cases, using the relational model. However, complex applications within e-science domains, require data models to properly reflect the real world entities similarly to their existence in reality. To properly model the Dublin Core elements and benefit from the utilization of the DC model, we address extension of the DC meta-data definition in terms of object-oriented modeling and object references. For instance, the object oriented modeling of the creator, publisher, contributor, and rights, better suits in scientific applications by allowing the creation of a more comprehensive relationships among objects. The object-oriented modeling of these elements allows their representation as real-world entities, and makes them related to each other via the means of relationship associations.

Figure 5.7 illustrates the design of an object-oriented database definition for Dublin Core schema elements that we introduce for use within the VL project, using the Unified Modeling Language [UML 98]. The Object Definition Language (ODL) schema for this meta-data is also provided as a subset of the enhanced object-oriented Dublin Core ODL schema definition, presented in Table 5.2. Users of Dublin Core can find more details and useful references related to the use of Dublin Core in [Hill 01].

![Diagram](image)

Figure 5.7: An Object-oriented schema for the Dublin Core meta-data

5.3.2.3 Enhanced Dublin Core Data Model

In order to satisfy the complex requirements for advanced scientific applications in terms of scientific data representation, additional meta-data vocabularies related to scientists and their performed experiments need to be added to the DC data model. As such, the extensions we introduce must allow to distinguish between raw and processed data, to specify the different kind of processes used for data manipulation, to indicate the devices from which data is being collected, and to support the reviewing for scientific experiment results.
Figure 5.8 illustrates a generic object-oriented schema representation for the VL data archiving. This schema is based on and enhances the DC meta-data model presented in Figure 5.7. The extensions to this schema first allow the distinction between raw and processed data, and second provide facilities for reviewer’s comments, processing methods definition, and specifications about the devices generating this data.

The extended part of the DC schema in Figure 5.8 presents the necessary additional entities required for the VL experimental environment. For instance, a Process is defined by a name, a description, and a set of parameters; it uses one or several raw data as input, and produces a set of processed data. It is possible that one or more persons review the produced data. The DC schema elements are related to each other through a number of relationships describing the associations between these entities. The multiplicity range (cardinalities) of the relationship associations define constraints on the data, a process for instance must have at least one input (raw data) and produce one or more output results (processed data).

![Enhanced DC Schema](image)

Figure 5.8: Enhanced Object-oriented schema definition for the VL archiving environment based on the Dublin Core standard

Table 5.2 presents the ODL schema definition for the enhanced object-oriented DC metadata, illustrated in Figure 5.8. The provision of the ODL definition presents the advantage of direct loads/creation of the DC schema into any database that is ODMG compliant. The use of enhanced Dublin Core metadata for modeling scientific applications has proven its
validity via its deployment, as a base, for the data definition within different applications, which are being developed within the VL information management framework. Among these scientific applications, we enumerate a generic model for experimental environment [KAH01], DNA micro-array and gene expression [KAB+01], Mass Spectrometry metadata analysis [EAG+01], and information management for material science applications [FAE+01].

<table>
<thead>
<tr>
<th>//Extended Dublin Core ODL Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface ArchivedElement : persistent {</td>
</tr>
<tr>
<td>attribute String Title = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Subject = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-full-text-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Description = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-full-text-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Type = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-full-text-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Format = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Identifier = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Source = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Language = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Relation = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Coverage = &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;Entity&gt; Creator</td>
</tr>
<tr>
<td>inverse Entity::CreatorOf;</td>
</tr>
<tr>
<td>relationship List&lt;Entity&gt; Publisher</td>
</tr>
<tr>
<td>inverse Entity::PublisherOf;</td>
</tr>
<tr>
<td>relationship List&lt;Entity&gt; Contributor</td>
</tr>
<tr>
<td>inverse Entity::ContributorOf;</td>
</tr>
<tr>
<td>relationship List&lt;Right&gt; Rights</td>
</tr>
<tr>
<td>inverse Right::RightsFor;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Right : persistent {</td>
</tr>
<tr>
<td>attribute String IPR = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Copyright = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Property = &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;ArchivedElement&gt; RightsFor[1,-1]</td>
</tr>
<tr>
<td>inverse ArchivedElement::Rights;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Entity : persistent {</td>
</tr>
<tr>
<td>attribute String Phone = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Fax = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Email = &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;Address&gt; EntityAddress[0,1]</td>
</tr>
<tr>
<td>inverse Address::AddressOfEntity;</td>
</tr>
<tr>
<td>relationship List&lt;ArchivedElement&gt; ContributorOf</td>
</tr>
<tr>
<td>inverse ArchivedElement::Contributor;</td>
</tr>
<tr>
<td>relationship List&lt;ArchivedElement&gt; PublisherOf</td>
</tr>
<tr>
<td>inverse ArchivedElement::Publisher;</td>
</tr>
<tr>
<td>relationship List&lt;ArchivedElement&gt; CreatorOf</td>
</tr>
<tr>
<td>inverse ArchivedElement::Creator;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Person : Entity : persistent {</td>
</tr>
<tr>
<td>attribute String LastName = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String FirstName = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Title = &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;Review&gt; Reviews[0,-1]</td>
</tr>
<tr>
<td>inverse Review::ReviewedBy;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Organization : Entity : persistent {</td>
</tr>
<tr>
<td>attribute String Name = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String ActivityType = &quot;&quot;;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Address : persistent {</td>
</tr>
<tr>
<td>attribute String Street = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Number = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute Short PostCode ;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String City = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String State = &quot;&quot;;</td>
</tr>
<tr>
<td>attribute String Country = &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;Entity&gt; AddressOfEntity</td>
</tr>
<tr>
<td>inverse Entity::EntityAddress;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>//Dublin Core Meta Data Extention</td>
</tr>
<tr>
<td>------------------------------------</td>
</tr>
<tr>
<td>interface Review : persistent {</td>
</tr>
<tr>
<td>attribute String Relation = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute Date Date;</td>
</tr>
<tr>
<td>relationship List&lt;Person&gt; ReviewedBy[0,-1]</td>
</tr>
<tr>
<td>inverse Person::Reviews;</td>
</tr>
<tr>
<td>relationship List&lt;ProcessedData&gt; ReviewFor[1,-1]</td>
</tr>
<tr>
<td>inverse ProcessedData::Preview;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface ProcessedData: ArchivedElement : persistent {</td>
</tr>
<tr>
<td>attribute Binary Content;</td>
</tr>
<tr>
<td>attribute String Comment;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute Date Date;</td>
</tr>
<tr>
<td>relationship List&lt;Process&gt; Uses[1,1]</td>
</tr>
<tr>
<td>inverse Process::Uses;</td>
</tr>
<tr>
<td>relationship List&lt;ProcessedData&gt; UsedFor[1,-1]</td>
</tr>
<tr>
<td>inverse ProcessedData::ProcessedDataUsedIn;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface RawData : ArchivedElement : persistent {</td>
</tr>
<tr>
<td>attribute Binary Content;</td>
</tr>
<tr>
<td>attribute String Comment;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;RawData&gt; Uses[1,-1]</td>
</tr>
<tr>
<td>inverse RawData::Uses;</td>
</tr>
<tr>
<td>};</td>
</tr>
<tr>
<td>interface Device : persistent {</td>
</tr>
<tr>
<td>attribute String Name = &quot;&quot;;</td>
</tr>
<tr>
<td>mt_make_entry &quot;make-entry&quot;;</td>
</tr>
<tr>
<td>attribute String Description &quot;&quot;;</td>
</tr>
<tr>
<td>relationship List&lt;RawData&gt; Generates</td>
</tr>
<tr>
<td>inverse RawData::GeneratedFrom;</td>
</tr>
<tr>
<td>};</td>
</tr>
</tbody>
</table>

Table 5.2: Enhanced ODL schema for the VL archiving environment based on the Dublin Core standard
5.4 Universal Database Access - Based on Standards

The work presented in this section describes the universal database access interface (called UDBA [Ben 00a]). The UDBA is a web-based framework, which is achieved through a set of functions for data and meta-data manipulation, that are supported by the combination of database technologies with current standard tools for data access such as ODBC and JDBC. The provision of these functions for accessing the internal structure of a database (data and schema) facilitates the process of creating intelligent web-based and non web-based applications. Therefore, universal database access is achieved via the development of generic tools through which users will be able to access several data sources regardless of their internal structure, data types, and location.

The development of universal database access interface brings three main advantages to the work in the area of information management and interoperation.

1. First, it presents a flexible interface to easily manage the database schema and objects through a web environment to which ordinary users are quite familiar.

2. Second, it allows to better explore the advanced features supported by the object-oriented/object-relational DBMSs and provides means to improve some of these features supported by their object database connectivity mechanisms (e.g. class inheritance, object identifier, and cross-reference relationships).

3. Third, it extends the implementation of the data types as required by complex scientific applications such as the multi-media information, large data sets, and complex interlinked objects.

The implementation of the universal database access interface is based on the following software technologies:

- Matisse object-relational DBMS [Mt 01] for schema representation and data storage.
- Active Server Pages programming model [ASP 01], which allows dynamic and interactive Web pages to be generated on the fly from the Web server.
- ActiveX Data Objects [ADO 01] programming extension for database connectivity. The primary benefits of ADO are ease of use, high speed, and low memory overhead.
- Matisse ODBC driver for accessing the heterogeneous databases, structure and objects, via a common interface
- JavaScript and VBScript languages, which overcome the limitations of HTML and allow the creation of functions, embedded within HTML code.

Using the universal data access interface, users do not need to know much about the data structure of the underlying data source. Users are only asked to specify the data source’s name. Then the application connects the user to the specified database, reads the schema structure, and automatically presents to the user, in a very flexible manner, a set of concepts, through which he/she can freely navigate and explore the database structure (schema) and its instances (data).

For the VL information management system, the universal database interface provide a facility similar to what the database vendors deliver as a “client interface” to manage the data and the schema for the database that runs on the DBMS server. The main difference is that the universal database access interface can be considered as a complementary tool
to the specific database interface that can also be accessed by different users through the Internet. Therefore, universal database interfaces brings to the DBMS server the advantage of: (1) being able to run on a web server environment, which makes it platform independent and widely used, and (2) since the UDBA interface is built on top of the middleware and standard solutions, it can be considered as a generic framework that can interface many databases that are compliant to standards.

![Data Model Diagram]

Figure 5.9: Example simplified Data Model for Authors and Publications

Figure 5.10 presents a screen shot of the universal database access interface and illustrates the four main components of this interface, namely, the **Database Connection**, the **Query Execution**, the **Results Presentation**, and the **Object Creation** modules. The example, shown in this section, is based on a database for **Authors** and **Publications**, for which a simplified model is presented in Figure 5.9.

As depicted in Figure 5.10, the implementation of the UDBA Web interface utilizes the frames mechanism. Therefore, the interface consists of four browsing areas (frames), each handling a set of sub-tasks of the system. The use of frames in this context allows the proper representation of complex objects and facilitates the navigation among those objects that are related to each other. The four frames of the UDBA interface consist of (1) database connection (at the top left), (2) schema exploration and query formulation (at the top right), (3) results presentation (at the bottom left), and new objects creation (at the bottom right).

The general steps for information access through the universal database access, pointed to in the figure by circled numbers 1 to 4, are briefly described below. Further details of the functionalities provided by the universal data access are described in section 5.4.5.

1. First, the user has to specify a data source name, and press the **Connect** button. The interface then connects to the corresponding database, reads its internal structure, and presents to the user a friendly interface. The database **Connection Module** handles this step, and provides the user with the complete structure of the underlying database through which he/she can freely navigate among the database components (tables/classes), select the desired items (or specify a query), and specify his/her desired output format for the results.

2. Second, the user has either to select a class/table name or specify an SQL query to be submitted to the data source server, choose his/her preferred output format for the
results, and press the *Query Execution* button. The *Query Execution* module extracts user inputs, checks for syntax and semantic errors, and builds the corresponding query to be sent to the database.

3. Third, the query results are returned to the user. The *Result Presentation* module allows the presentation of the query results to the user according to his/her desired format. The output format can be HTML, in the form of a table, or using a standard data exchange format such as XML and OIF. At this stage, the user can also choose to add/create a new object for the selected data type into the database.

4. Finally, the user has the possibility to create new objects in the database. The *Object Creation* module provides a flexible facility to dynamically create objects and store them into the database. This facility is provided to the user within the data representation module.

![Universal Database Access Interface](image)

Figure 5.10: Universal Database Access Interface

Please notice that for every step, there is a module supporting its functionality. These modules that constitute the universal database access framework are further described in more details within the following sections.
5.4.1 Database Connection Module

As depicted in frame (1) of Figure 5.10, the connection to a database server is simply done by specifying a database name and requesting the connection\textsuperscript{12}. When activating the connection module, this later automatically connects to the specified database, reads the structure of its schema, and organizes it in a simple format for the user. The format consists of:

- A pull-down scroll menu consisting of all the database classes, from which, the user can choose one class to browse or manipulate its instances.
- An input text area, where the user can specify an SQL statement to be submitted to the database. The SQL statement can be a select query, an update statement, an insert or a delete command.
- A pull-down list, consisting of all possible output representation formats for the selected class objects.
- The query execution button through which, the user validates and activates the query execution process.

5.4.2 Query Execution Module

The query execution module extracts the user input/selection, checks for errors, builds the corresponding query according to the user request, and launches the query execution process. The query execution module performs according to the following strategy:

- If an SQL statement is specified, it executes it without checking the class name in the pull-down menu
- If a database class/table name is selected, the query execution module reads the selected class name from the pull-down scroll menu, constructs the proper select query, for instance `select OID, * from <class name>`, and performs the query execution.
- If neither of the two inputs is specified, the query execution module prints a warning message asking the user to either select a class name or specify an SQL statement.

5.4.3 Results Presentation Module

The result presentation module extracts the corresponding output information for the user's requests and organizes the query results according to the specified format. Different layout possibilities are illustrated for presenting the information to the user. The user can choose either an HTML presentation, an XML format, an Object Interchange Format (OIF), or a table format presentation.

The result presentation module builds a dynamic output for presenting information to the end user. The presented output is dynamic in terms of columns and lines; and is based respectively on the selected attributes, and on the objects available for the chosen class.

Frame (3) in Figure 5.10 illustrates the output format for publications and authors using an HTML format. While, another example is presented in frames (3) and (4), which show on the left hand side the ArchivedElements with the XML format, and on the right hand

\textsuperscript{12}Specification for the underlying database name, location, and access mechanisms are supported and provided via the ODBC data sources facilities.
side the IPRs\footnote{ArchivedElement and IPR concepts are defined in section 5.3.2.3} related data using the table output format. Query results are represented on the screen and one page at a time, through which the user can navigate and scroll forward and backward.

### 5.4.4 Object Creation Module

In addition to the information retrieval and data representation, the dynamic creation of new instances for these classes that are being explored, can be achieved through the Add New<object> button, which generates the corresponding form for the data input and loads it into the database.

The object creation module allows the generation of dynamic forms for input to be filled up by users, to create new instances. Input forms are based on the structure of the current class that is being explored. Frame (4) in Figure 5.10, for instance, presents an input form for creating a new instance of the class Paper.

After filling the input form, the data specified by the user will be submitted to the database in order to create a new object (instance) for the considered class. Submission of the input data is accomplished by the Commit Event module, which dynamically reads the user inputs, builds the SQL command, and inserts the new object into the database.

In case of any input data errors (e.g. wrong type, invalid range), these errors will be identified by the system and reported to the user. Therefore, the updates to the database are not committed unless the user fixes all errors. At this level, providing the user with the possibility to go one screen backward and correct his/her errors (without losing any data), facilitates the objects creation process.

### 5.4.5 Further Benefits

In addition to the various advantages of the UDBA framework as presented within the previous sections, in the following sub-sections, we will provide more details and exemplify the three concepts that illustrate the strength simplicity, openness, and flexibility of the universal database access framework.

#### 5.4.5.1 Dynamic Query Definition and Results Presentation

The universal database access framework presents a dynamic facility for query definition and results presentation. The flexibility of the framework is made possible through the run-time access to data and schema structures of the underlying data source.

The flexibility in data representation is supported in terms of database structure characterization, object instances creation, and layout for the output results formatting.

- Depending on the defined attributes within each class, the user can select a set of properties and constraints to be considered for the query execution.
- Through the SQL statement, the user can also restrict the objects to be retrieved, via the specification of the condition predicates.
- According to the user specifications for the layout in data presentation, the results of a query are formatted to fit the selected formatting template.
The flexibility for query specification and conditions definitions are provided through the support of SQL commands for INSERT, UPDATE, DELETE, as well as for database schema manipulation. The following examples present in more details these features through a set of examples from the 'Authors and Publications' data model defined in Figure 5.9:

- Creating new instances for classes defined within the database schema are supported through the INSERT command. For instance, the statement: 
  ```sql
  INSERT INTO Paper (Title, ConfName) VALUES ('MegaStore', 'DEXA 2000')
  ```
  creates a new instance of the class Paper and assigns the values MegaStore and 'DEXA 2000' respectively to the attributes Title and ConfName.

- Updates to the database are supported through the UPDATE query statement. For instance, the SQL statement: 
  ```sql
  UPDATE Paper SET (Pages, ConfLocation) VALUES ('869-878', 'London, UK') WHERE title LIKE 'MegaStore'
  ```
  assigns new values '869-878' and 'London, UK' for respectively Pages and ConfLocation of the class Paper, when the condition title "LIKE 'MegaStore'" is satisfied.

- Removing objects from the database is supported through the DELETE command. For instance, the simple delete query: 
  ```sql
  DELETE Paper where Title = ' '
  ```
  removes all instances of the class Paper for which the Title is empty, thus, it can be used for instance to clean the database from some incomplete data.

- Exploring the structure of the underlying database is supported through a set of concepts for schema manipulation and is implemented using the Matisse DBMS.

### 5.4.5.2 Multi-level Navigation Through Relationships

Complex Objects in scientific and system engineering applications are characterised by a set of relationship properties that link related objects together and form a network of entities. A good example for complex inter-linked entities is the conceptual schema for the Virtual Laboratory archiving environment, presented in section 5.3.2.3. In this environment, for instance, the entity ArchivedElement is linked to several entities such as Publisher, Creator, and Right. These entities are inter-linked though the concept of relationship definition.

During a user session and along his/her navigation, the universal database access interface automatically determines all the links among the objects in the database and provides the possibility of navigational access among them. An example that illustrates the navigational facility provided by the framework is presented in Figure 5.13 (frames 3 and 4), in which the presented results are also augmented with the relationship links from each object to all other objects that are related to it. For instance the possibility to navigating from an ArchivedElement object and explore other objects related to it (e.g. Right, Contributor, Creator, and Publisher). These reference links allow the user to easily navigate through the database objects and freely explore the complete scientific data sets as they exist in real world.

The navigational mechanism of the system is implemented via the relationship capability offered by the Matisse DBMS\(^{14}\). This mechanism provides fast and direct access among inter-linked objects; thus, it eases the user interaction and navigation through the database objects.

---

\(^{14}\) In addition to its support for SQL ANSI standards, the Matisse ODBC drive provides extra features related to object-orientation and multi-media data manipulation.
5.4.5.3 Multi-Task Modules

The universal database access interface is an advanced Web framework that provides a flexible interface based on the knowledge acquired from the database structure, object instances characterization, and the output results presentation.

The development of the UDBA framework, presented above, benefits from modular design and implementation. Its complete development consists only of four modules that perform all the tasks required for the schema and data manipulation. The multi-task functionality of each of these modules is supported via a set of parameters, which are dynamically adjusted based on the data characterization and users’ specifications. The Result Presentation module for instance, is responsible for representing in a simple and flexible way any type of data generated either by a select query or an update statement, according to the user’s specifications and preferences. Similarly, the Object Creation module is a flexible component, which generates correspondent input data forms for classes defined within the database schema, and builds the proper insert query to be sent to the database, regardless of the structure of the information and the size of data that it contains.

5.5 Data Access Security and Information Visibility (Safe/Reliable Data Export)

The work described in this section concentrates on designing and implementing an interface to support role-based access rights facility, allowing organizations to share part of their data with certain other organizations. More details concerning a full description regarding data access control principles and mechanisms is outside the scope of this thesis, the reader can refer to the various approaches that have been published over the past few years. Among the published proposals, [SS 94] addresses principles related to access control. [Bald 90] focuses on naming/grouping privileges to simplify security management in large databases and [SC 96, SF 94, Tho 91] proposed access control extension mechanisms to support role-based models. Other publications concentrate on the applicability of a role-based data access control to solve security problems in Intranet environment [TC 97], and to support information access rights and visibility levels in Virtual Enterprises [FAG+00, GAH 01].

The safe/reliable data export framework is a user-friendly web interface to access and retrieve a wide variety of data, based on predefined export schemas. The access to data, through those export schemas, defined on top of existing databases, is achieved via a role-based access control.

The implementation of the safe/reliable data export interface is based on the universal database access framework presented in section 5.4, and extends it to support access rights and visibility levels to a subset of information, based on export views definition and user’s (organization’s) roles assignment. Roles are used to simplify the description of allowed access characteristics to database objects. The approach for granting user permission to access a subset of the information within each application is achieved through role assignment.

The scope of access to database objects using the Safe/Reliable data export is achieved through the following steps:

- Create roles according to the job functions (e.g. within the organizations).

---

15Export schemas definition within this section refers to views definition augmented with the concept of role-based access control.
• Grant users permissions (access authorization) based on these roles.
• Define export views based on database objects, and
• Assign the defined export views to the roles on the basis of their responsibilities.

The remaining two sections illustrate the adaptation of a role-based access control to existing applications, in which section 5.5.1 addresses a role-based access control definition, a facility helping the application developer in defining the necessary mechanisms for creating a role-based access control; while section 5.5.2 provides an interface for data publishing based on predefined export views and users authentication.

5.5.1 Role-based Access Control Definition

Figure 5.11 illustrates the schema definition for the meta-data characterising the role-based access control to export views. This schema is defined for the implementation of the safe/reliable data export interface. As described later in this section, when necessary, this generic data model will be augmented to different database descriptions, within several applications of the VL, in order to control and restrict external access to their data. In order to support the database administrator with the creation of export views based on the fusion of several database objects, a class named Element is defined to cope with this issue, thus, the designed model supports the definition of export views based on several joint classes.

Figure 5.11: Schema Definition of the role-based Access Control with Export Views

The implementation of the safe/reliable data export framework is supported via the development of related tools to create export views, define roles, and assign users to those roles based on their privileges and access rights. As an example of these tools, we present in Figure 5.12 a screen shot of a user-friendly interface to create export views (similar interfaces are also available for defining users permissions and roles). This example also illustrates the steps involved in defining export views, including:

1. The database administrator (DBA user) connects to a given data source by simply specifying the database name and pressing the Connect button.
2. DBA user selects a class name from a list provided by the system. The list corresponds to the set of classes extracted from the structure of the underlying data source.
3. DBA user proceeds with the export view creation, by selecting a set of attributes and specifying optional condition predicates.

![Figure 5.12: Interface for Views Definition](image)

Once the user presses the SubmitView button, the corresponding commands (statements) for the view creation will be built on the fly and committed to the specified database. Two specific cases are supported here:

**1st Case**: If the underlying database system supports the definition of views (e.g. Oracle DBMS), an SQL command is formulated and executed, creating the export view. In this case the corresponding SQL statement will be:

```
Create View Archives_V1 as
Select Title, Description, Date, Language, Coverage
From ArchivedElement
Where (Date > '03/01/2002' and Format = 'NCDF').
```

**2nd Case**: If the underlying database system does not support the built-in components for the definition of views and roles (e.g. Matisse DBMS), as it is the case for most object-oriented database systems, the database schema for the corresponding application must be augmented with the data model of the role-based data access control. In this case the following specifications are realized for the elements of the export view defined in Figure 5.12:
• **Select Specification**: Select Title, Description, Date, Language, Coverage
• **From Specification**: From ArchivedElement
• **Where Specification**: Where (Date > '03/01/2002' and Format = 'NCDF')

Other interfaces are also developed for the purpose of creating export schemas, including user definition, and the roles assignment for users and groups. In terms of flexibility and user-friendly facilities, the implementation of these interfaces is similar to the one for export view definition.

### 5.5.2 Flexible Role-based Access Interface

The role-based access interface, primarily developed for data publishing to outside users, relies on safe and reliable mechanisms for data export and publishing. The interface is safe and reliable in the sense that it is based on the definition of roles, defined at different access levels. A VL user must be authenticated by the system before any attempt to access information. In addition, users are not aware about the complete structure of the underlying database, each user only sees the part of the database for which he/she has gained the proper access rights.

![Safe/Reliable Data Export Interface](image)

Figure 5.13: Safe/Reliable Data Export Interface

Figure 5.13 illustrates an example of the safe/reliable data export interface for an external user of VL connected through the Web. The interface tool, first checks and authenticates the user connection for the selected data source, in this case the VL archive database (following the example of figure 5.12), and only then, it provides the user with a set of information (in
this case the class names including the ArchivedElement] based on his granted access rights and visibility level. Therefore, the "user" access to the database objects is restricted via the assigned roles, and users that do not gain permissions to the data source are prohibited from any access to those objects.

Once the user is authenticated for accessing a selected data source, a connection will be established to the underlying database, and the system will provide the user with some interaction facilities to those database objects for which access permissions are granted. From the safe/reliable data export interface, the user can perform the following tasks:

- Select a class/table name and browse or retrieve its instances. At this level, the user can also specify an SQL query to be executed and evaluated against the underlying data source.
- Choose an output format for the data presentation, currently the framework supports the following formats: HTML, XML, OIF, and table format.
- Submit a query for execution on the database, and receive back the results according to the specified format.

The Query Execution module checks the user input, creates the appropriate query, restricted to his/her export view, to be sent to the database, and launches the Results Presentation module, which takes into consideration the output format specified by the user.

In addition to presenting the results on the screen, the framework also provides its users with the following possibilities (Figure 5.13. frame (3)): 

- Possibility to Upload the query results and save them locally for future use and examination. The results are stored according to the format chosen by the user.
- Possibility to Navigate among the database objects through the defined links and relationships to other objects. Each object is augmented with a set of links referencing all the objects to which it is related.

5.6 Physical Database Performance Analysis

Nowadays in all organizations large or small, databases constitute the most critical elements, as the brain handling the organization's information. They ease the storage and retrieval of massive amount of information, and provide the proper facility for multi-user information sharing and concurrent access control. DBMSs provide major facilities to the application developers, thereby, they are also considered as an important factor on determining the application performance in terms of data storage and information retrieval.

A main approach in this direction is to address the issue of scientific database system's performance, especially for storage/retrieval of large binary objects within the DBMS itself. In order to achieve the most efficient physical implementation for VL, the main focus within this section will then be to address some benchmarking tests at the physical level of the Matisse DBMS, which is currently used within the VL project. Therefore, this section addresses the results of the physical database performance analysis regarding the manipulation of large scientific data sets, when using the object-oriented Matisse database system as the base for implementation.

In order to properly conduct these data access performance tests at the physical level of Matisse DBMS, first we have designed and developed of a set of basic functions to access
large database objects within the Matisse database. Second we have performed some tests to evaluate and analyze performance of the Matisse database when storing and retrieving large binary objects.

5.6.1 Specific Functions to Access Binary Large Objects (Blobs)

A set of specific functions to read/write large objects (blobs) from/to Matisse database are defined and implemented. These functions facilitate the access to large objects, which are directly stored/retrieved into/from the Matisse database as a list of binary bytes. Four functions are developed for the management of large binary objects [Ben 00b]. Each of these specific functions is implemented based on a set of elementary functions, provided by the Matisse C++ API interface.

- **ConnectDatabase**: Connects to a distributed database that runs on a remote host.
- **LoadBlob**: Loads a binary large object into the database as the value for the attribute mentioned as defined within the class.
- **ReadBlob**: Reads a binary object using the entry from the attribute defined within the class.
- **DisconnectDatabase**: Disconnects from the database on host.

These functions hide the programming complexities from the user when accessing binary objects, thus, provide easy and simple access for the user. These functions are also used for the implementation of the benchmarking tests, described in the next section.

5.6.2 Benchmarking Tests For Matisse Database System

As a part of the design of most efficient physical implementation for the VL project, a set of benchmarking tests are performed on the Matisse DBMS to evaluate the performance of different implementation approaches for storing/retrieving very large binary objects in the database. Large real data sets from the application case of VISE, focused on visualization and simulation functionalities of the VL project (described in section 5.2), are used as input for the benchmarking tests. These tests are performed for read and write accesses using the functions described in section 5.6.1 and for two different implementation-configurations in Matisse. Namely, first a Matisse database that uses a normal disk (regular file managed by the system), and second using a raw disk partition (managed by the DBMS itself). For this benchmarking of the Matisse database system, the following test case input was designed and applied:

- 100 objects are used for both purposes of storage and retrieval accesses,
- The size of these objects range between 73 Kilobytes and 12 Megabytes,
- The total size of the 100 objects (loaded into the database) is around 570 Megabytes,
- The objects’ storage/retrieval starts with object number one (which is of the smallest size: 73 KB) and gradually increases in size till object number 100 (which is of the largest size: 12 MB).
• The difference in size between every two loaded objects $N$ and $N+1$ is approximately 70 Kilobytes, where $N = 1 .. 99$.

• The database software runs on the Arches machine at the University of Amsterdam, which consists of dual Pentium II processors with 512 MB memory, 9 GB disk storage, and supports several network communications (Fast-Ethernet, Myrinet, and Gigabit Ethernet).

• To reach the best physical performance, the database configuration, which is chosen based on the available hardware for VL and also considering the suggestions from the Matisse DBMS developers, is as follows:
  
  - Database silo size (total storage capacity): 2 Gigabytes
  - Database Bucket size: 64 ($64 \times 512 = 32$ KB)
  - Database cache size: 4000 ($4000 \times 32$ K = 128 Megabytes)

These tests are very important in the sense that they allow us to determine the best data access performance that can be reached when storing large objects within the DBMS.

Figure 5.14 illustrates the Matisse database performance when storing/retrieving the 100 objects, of different sizes, one by one. The X-axis values represent the size of the individual objects being stored/retrieved, each illustrated by its size expressed in Megabytes. The Y-axis values represent the database access time for both storing (write) and retrieving (read) of each object, expressed in seconds. More details concerning the input/output data used for the benchmarking tests are given in [Ben 00b].

For the first case, in which the database uses regular file system for the data storage, the average data access time for the write operation reaches 664 KB per second and for the read operation reaches 784 KB per second.
For the second case, in which the database uses the raw disk partition for the data storage, the average data access time for the write operation reaches 3.35 MB per second and the read operation reaches 4.92 MB per second.

5.6.3 Observations

From the benchmarking tests performed, according to the chosen configuration parameters for the test case described in previous section, we conclude the following observations:

**Observ 1:** On the average, for the case of using regular file system and for objects that are smaller than 3 megabytes, the performance of Matisse database system is almost the same for both read and write operations. However, for larger objects, the difference between read and write operations is rapidly increasing.

**Observ 2:** For the case of using the raw disk partition, both read and write operations almost take the same amount of time for objects up to 6 megabytes. For larger objects, the difference between the read and write operations has a minor increase.

**Observ 3:** On the average, the Matisse database system performance when storing/retrieving large objects improves up to 6 times with the use of the raw disk partition.

**Observ 4:** The database access time is more regular when using raw disk partition than in the case of using normal file system, which is probably influenced by other I/O performed by the operating system.

5.6.4 Lessons Learned

To better support the complex requirements of scientific applications in VL, and to design the most efficient physical implementation for the database, some data storage/retrieval benchmarking tests are necessary to be performed. The test cases must be carefully designed to help with the evaluation of the approaches for physical database design, while considering the performance of the DBMS itself for managing large scientific multi media data sets. Besides the tests performed above, additional tests may become necessary for instance to consider the hardware configuration in terms of (1) using several raw disk partitions, (2) increasing the database cache size, and (3) using powerful machines, mainly dedicated to run the database server for large applications.

The knowledge gained from benchmarking plays an important role in defining the physical database design and specific data storage strategy for each application, based on the data input/output requirements. For instance, considering the performance results achieved above, a small to medium size application that does not require more than 5 MB per second for its input/output, can follow an archiving strategy where the large data sets are stored together with their meta-data in the same Matisse database. However, more complex applications that may require better performance or proper storage of larger data sets can benefit from a different strategy, where the database only holds a link to the large objects, that can in fact reside at the place where they are generated or heavily used.

5.7 Conclusion and Discussion

This chapter addressed concepts for building a robust application to efficiently manipulate large and complex scientific data sets in the VL environment and introduced flexible interfaces for data access in the VL. The addressed concepts, however, are not only specific to
the VL, rather, most concepts presented here also facilitate, for instance, the creation of general digital libraries for scientific applications and can support the manipulation of their large multimedia data sets.

In different sections of the chapter, we illustrated possible strategies for storing scientific data, and outlined the major benefits gained when considering standards during the modeling and implementation phases of advanced and complex applications. Such concepts can be applied to different applications and serve for improving their development life cycle from information classification strategies to modeling the constructs, and from the design of information access mechanisms to the analysis of performance criteria.

The concepts presented in this chapter extend and complement the work presented in previous chapters. The extensions addressed in this chapter focuses on the following:

- The combination of database standards and Web technologies, to facilitate the process of developing information integration mechanisms among networked applications.
- The use of universal data access mechanisms to provide common interfaces for accessing heterogeneous databases through a common set of code.
- Standard information modeling for scientific applications to provide a common understanding about the data and meta-data dealt with, in scientific applications.
- The considerations of scientific multimedia information, large data sets, and complex inter-linked objects.
- Provision of mechanisms and tools for scientific data publishing, based on tailored and restricted access on sharable data, while preserving systems' autonomy; and hiding private data from outside users.

5.7.1 Contribution to GFI_2S

The concepts introduced in this chapter addressed a number of important issues that are applied at every site to better enable it as a node within the cooperation network. The data storage strategies and the performance issues assure the database efficiency at the local sites. While, the definition of restricted schemas and the development of universal and schema free tools to access them, allow sites to share part of their data. The contribution of the approaches described in this chapter to the GFI_2S integration approach presented in Chapter 6 is two-fold:

- On one hand, since the GFI_2S integration architecture preserves autonomy for individual sites, those sites can benefit from the use of the important concepts developed for VL information management, in order to efficiently build their applications independently of other sites.
- On the other hand, the use of standards at individual sites as adopted for VL, especially for the information modeling and universal data access, helped the development of information integration mechanisms for GFI_2S, which was necessary to support collaboration among heterogeneous and autonomous sites.