Scientific Information Management in Collaborative Experimentation Environments

Kaletas, E.C.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3

A Framework for Study and Evaluation of Related Work

As outlined and exemplified in Chapter 1, emerging scientific experiments and applications have many challenging characteristics, and scientists face several difficulties when making advanced experiments. In addition, virtual laboratory (VL) has been proposed in Chapter 1 as a solution environment to support scientists with their experimentations.

Further elaboration on the emerging e-science experiments and applications is provided in Chapter 2. As such, Chapter 2 introduced a characterization of the proposed virtual laboratory solution, and presented the results of the use case analysis performed to identify different types of VL users and their activities within the VL. Chapter 2 also identified and described the needs and expectations of different VL users (i.e. the user requirements) as well as the requirements for the base VL infrastructure (i.e. the base ICT infrastructure requirements) to properly fulfill the user requirements.

A number of current research and development efforts are focused on the field of producing support environments for scientific experimentations. These efforts are mainly driven by two factors: First, the need for such support environments is increasing due to the availability of new experimentation techniques and instruments, which makes it possible to perform highly-complex experiments with challenging characteristics. Second, the growing availability of enabling technologies, standards and paradigms make the development of support environments feasible and efficient.

Considering the complexity involved in building a virtual laboratory and further in supporting its information management requirements, a strong background study of other related work seems a necessity. Therefore, on one hand in order to learn from the previous achievements, and on the other hand not to repeat and reinvent the wheel, a vast and detailed study was performed. This study included the related research on the enabling technologies, standards and paradigms, as well as a vast study of the state-of-the-art in related prototypes and systems. The latter study, on
the state-of-the-art, included some 28 developed systems.

Please note that a wide variety of systems have been surveyed, and not limited by the support environments mentioned in Chapter 1. As a result of this study, only a selection of the most significant and relevant systems to the research performed in the thesis are further refined in this chapter.

In order to document the results of this vast study on related work, the following structure is considered for this chapter (see also Figure 3.1):

In Section 3.1, first the results of the study on enabling technologies, standards, and paradigms are provided (left side of Figure 3.1). The related work on information models, information management approaches, resource management, and a few other related standards, technologies and tools are addressed in this section. Thus a wide spectrum of the base areas related to this dissertation, covering from ODMG and Dublin Core standards to federated information management, and from Grid and workflow management systems to virtual organization paradigm, are all briefly addressed.

The remaining of this chapter is however focused on the description and classification of a large number of related prototypes/systems (right side of Figure 3.1). As mentioned above, enabling technologies, standards and paradigms make the development of support environments feasible and efficient. In Figure 3.1, this relationship is represented with a block arrow from enabling technologies, standards and paradigms to related work. The defined classification addresses the main focus of the related system, being the orientation of the system towards experiment context, computation, instrumentation, or metadata. The studied related systems are all evaluated against a systematically defined set of criteria. The evaluation criteria encompasses the criteria defined as requirements in Chapter 2 and further extends with the focus on the functionality provided by the system, the applied enabling technologies, stan-

---

Figure 3.1: Study framework for related work
3.1 Related Enabling Technologies, Standards and Paradigms

In this section, some of the related (existing or emerging) technologies, standards, and paradigms are presented that represent the enablers for the development of supporting ICT infrastructures for scientific experimentations. Enabling technologies, standards and paradigms provide certain functionality, mechanisms, facilities, models, and methodologies that can be applied by ICT infrastructure developers to address some of the requirements for the support infrastructure (see Section 2.6). Application of these enabling technologies, standards and paradigms for developing support environments forms the base for one of the evaluation criteria defined and described in Section 3.2.

The enabling technologies, standards and paradigms presented in this section are categorized into four groups; namely information models and standards, distributed/federated information management systems, resource management technologies, and other related technologies, paradigms and tools (see Figure 3.2). The first group addresses the representational aspects of scientific information and includes related information models and standards (in Subsection 3.1.1). Standardization is among the most important constituents for interoperability. One specific need for interoperability rises when sharing and exchanging information among multiple sites. The second group of enablers focuses on distributed/federated information management (in Subsection 3.1.2), that provides generic mechanisms for sharing and exchange of information among multiple information centers, which may at the same time be autonomous and involve different types of heterogeneity (please refer to ‘Data Characterization’ in Section 2.2). Resource management technologies (in Subsection 3.1.3) constitute the third group of enablers. Experimentation involves utilization of several different resources. Efficient resource management is needed to ensure high availability and performance of resources as well as to ease their usage by scientists. Grid and Data Grid technologies as well as some other related technologies are presented in this area for the management of computing, networking and storage facilities. The last group focuses on enablers addressing other aspects of experimentation. Technologies and paradigms in this area include workflow management systems, virtual organization paradigm, and toolkits for Grid software development (in Subsection 3.1.4).

Each of these groups will be further elaborated on in the remaining of this section. For each group, brief information about representative systems will also be provided.
3.1.1 Information Models and Standards

Information models provide a means for describing entities and concepts involved in a domain in a structured way, whereas standards make it possible for collaborators to share and exchange information. The need for generic, flexible, extendible information models to represent complex experiments and information types was mentioned as one of the major requirements in Subsection 2.6.2. Availability of such models will enable the development of generic and uniform mechanisms for the management of scientific information represented by these models. Standards, on the other hand, further allow for the resolution of semantic, syntactic and systematic heterogeneities that may occur among the information models and information management systems of collaborators.

Three examples of information models and standards are provided here; namely the ODMG Standard, WebDAV Distributed Authoring and Versioning Protocol and Dublin Core Metadata Standard. Each of these standards addresses one of the different types of heterogeneities. ODMG Standard is at the level of an object store; and it addresses syntactic heterogeneity among multiple object stores by defining a standard data model for defining object schemas, a standard language for object querying and manipulation, and a standard format for object exchange. WebDAV protocol aims to overcome systematic heterogeneity by providing a Web-based environment and mechanisms for multiple users to publish, co-author and annotate documents. Finally, Dublin Core Metadata Standard is at the level of content, and it addresses semantic heterogeneity by standardizing the elements and their meanings in the descriptions of documents.

![Related Enabling Technologies, Standards and Paradigms](image)

Figure 3.2: Related enabling technologies, standards and paradigms
3.1. Related Enabling Technologies, Standards and Paradigms

**ODMG Standard.** The Object Data Management Group (ODMG) [76] defines the **ODMG Standard** for modelling of persistent objects [77]. Version 3.0 of this standard includes one data model component and two programming language-independent object specification languages. The *Data Model* component defines type specification and implementation with respect to objects, literals, properties, and operations. The *Object Definition Language* (ODL) is used to define the schema and operations of an object store. The *Object Query Language* (OQL), on the other hand, is an object manipulation language similar to SQL with extensions for object-oriented concepts like complex objects, object identity, path expressions, and operation invocation. OQL is compatible with the ODMG data model. Another component of the ODMG standard is the *Object Interchange Format* (OIF), which is used to dump the current state of an object store to a file and load it back from the file, as well as to exchange objects between object stores. ODMG also defines bindings for different programming languages, including Java, C++ and Smalltalk.

**WebDAV Distributed Authoring and Versioning Protocol.** WebDAV [78, 79] defines a set of extensions to HTTP. Its objective is to reach interoperability by allowing users to publish documents in the Web and to co-author published documents in-place. Documents can be of any type, from simple HTTP documents to software program codes [80]. To this end, WebDAV capabilities focus on three points: *Overwrite prevention* to prevent simultaneous modifications on a document and hence to maintain the document consistency; *properties* of published documents (i.e. metadata) in the form of name-value pairs, and facilities for creating, modifying, deleting and retrieving metadata, and for inter-linking documents through hypertext links; and finally *name space management* to manage the resources occupying the name space of a Web server (i.e. copy, move, delete documents).

**Dublin Core Metadata Standard.** The Dublin Core Metadata Initiative [81] is an open forum engaged in the development of the **Dublin Core Metadata Standard**. The standard contains an element set for describing a wide range of on-line resources. The standard is mainly developed for describing document-like objects, and contains elements for document metadata. However, the standard can also be applied to other sources if their metadata resembles a typical document metadata. The Dublin Core Metadata Standard comprises fifteen elements. These elements are grouped into three: The content-related elements include *coverage, description, type, relation, source, subject* and *title*. The elements related to intellectual property are *contributor, creator, publisher* and *rights*. The remaining set of elements is related to a particular instantiation of an item and includes *date, format, identifier* and *language*. Detailed descriptions of each of these elements can be found in the initiative Web pages [81].

### 3.1.2 Distributed/Federated Information Management

As emphasized by the examples of emerging experiments and applications presented in Section 1.1, scientists need to access multiple information sources during exper-
plementation, especially during the phases of experiment design and analysis of the results. The need is more obvious in fields like biology, where several related areas study different aspects of life and organisms, such as genetics and biodiversity. For example, a scientist working on metabolic pathways may need different types of information about a gene, such as expression profiles, sequence, or chromosomal location, in order to determine the role of that gene in a pathway. Currently, such information is stored in databases maintained by different organizations, without any standardization in modelling or representation. Generic distributed information management can be applied by supporting ICT infrastructures to address the user requirements for integrated access to multiple heterogenous information sources and for secure sharing and exchange of scientific information (see Section 2.5 for user requirements). Furthermore, by defining integrated schemas, scientists can transparently represent, persist and query complex relationships among heterogeneous pieces of scientific information.

Distributed information management addresses issues related to the management of geographically distributed information. A distributed database consists of a collection of logically inter-related databases [82]. The distribution is mainly for better performance reasons. A distributed database is typically designed as a whole and managed centrally [83].

In a fully federated database on the other hand, many databases contribute data and resources to a multi-database federation, while each participant has full local autonomy [83]. Federated information management [84, 85, 86, 87, 88, 89] can enable sharing and exchange of information among collaborating sites, while preserving the autonomy of the sites and ensuring the security of proprietary information. Federated access to data from diverse sources provides users the possibility of securely accessing all data made available by these sources, while hiding heterogeneity of the underlying platforms with respect to their data modelling approaches, data manipulation languages, and distinct schemas [86].

Figure 3.3 shows a classification of federated database systems as described in [86]. In loosely-coupled systems, every user in the system is responsible for developing the integration mechanisms suited to her/his requirements. Generally, there is no integrated schema in loosely coupled systems, and users directly access the information that they need from the respective data source and adapt it to their view of data. Tightly-coupled systems, on the other hand, provide common integrated schemas in a single database environment for all users, where accessing the local data sources are transparent to the users. In single-federations within tightly-coupled systems, there is only one global integrated schema visible to the federation members [85], while in multiple-federations every federation member generates its own integrated schema as its personal view of available information [84, 86, 90].

Other classifications for federated database systems can be found in [84] and in [91]. [82] is the well-known source for information about distributed database systems. [87] and [86] provide detailed information about federated/multidatabase systems, such as an overview of existing systems, issues like schema integration, query processing, and database languages. [92] discusses the heterogeneity and interoperability issues involved in information brokering through a metadata-based architecture.

Five systems are described below as examples of federated information manage-
ment systems. *PEER* is a multiple-federation tightly-coupled system, while *Donaji* is a single-federation tightly-coupled system. The remaining two systems, namely *DiscoveryLink* and *Virtuoso*, can be classified in between loosely-coupled systems and single-federations due to the fact that these systems do not support (semantic) integration of information, although they provide transparent access to multiple heterogeneous information sources. The fifth system described here, *Polar*\(^*\), is not a federated information management system itself. It is a distributed query processor that utilizes Grid facilities for execution of distributed queries.

**PEER.** The *PEER* system [93, 90] is designed and developed at the CO-IM Group [73] of the University of Amsterdam. *PEER* is an object-oriented multiple-federation system. Its architecture is based on a *global data model* and a *global language*. *PEER* introduces and develops federated schema management and federated query processing. In federated schema management, several schemas co-exist in every node in a *PEER* federation, including local, export, import, and integrated schemas. *Local schema* represents the information stored locally at the node. Every *export schema* covers a specific part of the local information that is defined to share with other nodes. Every *import schema* represents the export schemas of other nodes from which this node is authorized to import information. The *integrated schema* at every node is composed of inter-linking of the local schema with the import schemas. The integrated schema is finally and precisely defined at each node through attribute and relationship derivations in relation to its base schemas (i.e. the local and import schemas). Using the federated query processing, users can issue queries against any schema at the node. When a query arrives, it is converted into its source (derived) specifications, decomposed into sub-queries on those sources, then submitted to and evaluated against schemas at the sources, and finally the query results are returned and merged.

**Donaji.** *Donaji* [94] is a semantic framework for multi-databases. It is based on a single-federation tightly-coupled architecture. The metadata model in *Donaji* is based on the ODMG2.0 object model. The AQUA query algebra is used as the global query language. Components of the *Donaji* multi-database environment include the following: *Conceptual mediators* support access to the metadata and mappings

![Figure 3.3: Taxonomy of federated database systems](image-url)
between global and local sources, decomposition of global queries into local ones, and translation from global conceptual schema structures to local schema structures. *Query processor* is the logical unit for handling user's global queries. *Temporary object repository* is used during the decomposition into sub-queries and assembly of results from the local data sources. *Operational mediators* translate the sub-queries into the query languages of the local data sources and place the local query results into the temporary object repository, and resolve terminology conflicts and possible scope mismatches.

**DiscoveryLink.** IBM's *DiscoveryLink* [95] is a federated information management system originally developed for life sciences, although it is generic enough to be used also in other domains. It is developed using technologies from two other IBM products/projects, namely DataJoiner and Garlic project [96]. DiscoveryLink is a relational system supporting SQL3 standard. The studied version (version 7.2) supports read-only access to data sources. DiscoveryLink uses a wrapper-based architecture. Specific wrappers are developed for several relational database management systems (e.g. Oracle, SQL Server), and for file-based data sources (e.g. comma-separated/tab-delimited text files, MS Excel files). Structures in data sources are exposed to DiscoveryLink as tables, by importing their definitions into DiscoveryLink. There is no integrated schema; semantic integration task is left to the applications, which can be realized to some extent by defining views on the imported tables. When a query arrives on the imported data structures, DiscoveryLink server generates alternative query plans, which are evaluated against a cost model and capabilities of data sources. Then the query is decomposed into sub-queries according to the selected plan, each of which is executed by data sources.

**Virtuoso Universal Server.** *Virtuoso* [97] from OpenLink Software is a data middleware which provides transparent access to heterogeneous data resources. Virtuoso approach is similar to the one of DiscoveryLink; it imports table definitions from underlying data sources to Virtuoso server and allows users to issue queries against these imported table definitions. Virtuoso supports accessing data from Native Virtuoso SQL Database, different SQL databases (e.g. Oracle, SQL Server), and native XML databases. It provides different access mechanisms including Web services, SOAP, and HTTP.

**Polar*.** In [98] a prototype distributed query processing (DQP) system called *Polar* is described, which runs over the Globus Toolkit (described in Subsection 3.1.3). Polar uses the ODMG Data Model and OQL. It accesses external data sources through a set of wrappers, which provide the uniform interface and data types that Polar requires. During query processing, user queries specified in OQL are first parsed, a logical query plan is generated and optimized, and this logical query plan is transformed into a physical plan by selecting algorithms that implements each of the operations in the logical plan. The physical plan is then partitioned into a multi-node plan by inserting parallelization operations. In the final phase, machine resources are
allocated for each sub-plan by the scheduler. The information needed for resource allocation comes from the cost estimation performed by the query optimizer and from the Grid information services. The prototype implementation uses different Globus Toolkit services during distributed execution of queries, for instance, GSI for single sign-on, DUROC for synchronized process startup, and GASS for executable staging. All these features are accessed through the MPICH-G interface, the Grid-enabled message passing system.

3.1.3 Resource Management Technologies

Availability of high-performance infrastructure and sharing and management of resources provided by such an infrastructure were mentioned among the major requirements in Section 2.6 for the transfer, storage, processing and analysis of large data sets generated during experiments. In this subsection, related resource management technologies are presented and analyzed: Grid for the management of computing and networking resources; Data Grid for the management of storage resources; and some other resource management-related technologies for metacomputing and for distributed object management.

Grid Computing

In the late 90s, Grid computing [99, 100] was proposed as a distributed computing infrastructure for advanced sciences and engineering fields. Grid computing enables coordinated and dynamic resource sharing for collaborative problem solving in large scale distributed computing environments. Although a relatively recent technology, Grid is the de facto distributed resource management technology today. It provides mechanisms and develops standards for security, allocation and monitoring of resources, and submission, monitoring and management of remote jobs on these resources. Next generation of Grid architecture is based on the Open Grid Services Architecture (OGSA) [101]. OGSA builds on concepts and technologies from the Grid and Web services communities. It supports Web Services Description Language (WSDL) [102] and defines a uniform exposed service semantics (the Grid service). Furthermore, OGSA defines standard mechanisms for creating, naming and discovering transient Grid service instances, provides location transparency and multiple protocol bindings for service instances, and supports integration with underlying native platform facilities.

Globus Toolkit. Globus Toolkit [72] is a reference implementation of Grid concepts and is considered as the de facto standard for today's Grid computing. The Globus Toolkit employs a "bag of services" approach. Users can select the set of tools that they need for their specific requirements during their application development. In this approach, the "bag" can also be extended with additional tools. The toolkit includes resource management services such as resource discovery (using LDAP based catalogues), allocation, and monitoring [103, 104, 105]. The Globus Toolkit also implements the Grid Security Infrastructure (GSI) defined by the Grid computing, which
describes authentication techniques in wide area networks [106, 107].

Data Grid

Data Grid aims to define an integrated architecture that allows coordinated application of different technologies to data-intensive application domains. The name Data Grid emphasizes its role as a specialization and extension of the Grid. Research efforts related to Data Grid is mainly organized under two groups. The first one is the Database Access and Integration Services Working Group of the Global Grid Forum (GGF-DAIS) [108], and the second one is the UK e-Science Programme Database Task Force [109]. The current efforts in this field mostly concentrate on identifying the requirements and proposing mechanisms for accessing storage facilities from the Grid environment. The goal of The Data Grid work described in [110] is to define the requirements that a Data Grid must satisfy, and the components and APIs that will be required in its implementation. In this direction, a layered architecture is designed. The bottom layer corresponds to core services, being either Data Grid specific services or generic Grid services. The higher layer Data Grid services include replica management and replica selection. Core Data Grid services are data access and metadata access services. Other Data Grid services include an authorization and authentication infrastructure, resource reservation and co-allocation mechanisms, performance measurements, and instrumentation services. Most of the proposed mechanisms follow the Web services approach, specifically targeting integration of storage facilities within the OGSA framework [101].

The studied related work for Data Grid includes the European DataGrid Project, Open Grid Services Architecture Database Access and Integration Project, Database Access and Integration Services, Spitfire, Replica Manager, and GridFTP.

European DataGrid Project. The European DataGrid [111] is a project funded by the European Union with the aim of setting up a computational and data-intensive grid of resources for the analysis of data coming from scientific exploration. The project addresses sharing of huge amounts of distributed data over the currently available network infrastructure, i.e. emerging computational Grid technologies. The DataGrid project is divided into twelve workpackages, one being the Data Management Workpackage involved in defining and implementing the middleware component of the project. The goal of the Data Management Workpackage is to specify, develop, integrate and test tools and middleware infrastructure that will allow securely accessing massive amounts of data in a universal global name space, moving and replicating data, and managing replicas. Replica Management is the largest task in this workpackage. Another task, the Metadata Management Task, aims to provide transparent, secure access to metadata for Grid middleware and applications (please refer to Spitfire that is described later).

OGSA-DAI. The Open Grid Services Architecture Database Access and Integration Project (OGSA-DAI) [112] is concerned with constructing middleware to assist users with access and integration of data from separate data sources via
the Grid. It targets identifying the requirements, designing solutions, and delivering software that will meet this purpose. The first limited functionality release of the OGSA-DAI software is available from [112], and includes GridDataService, GridDataServiceFactory and GridDataServiceRegistry components as well as example client components that allow users to connect to MySQL and Xindice databases from the Grid environment.

**DAIS.** The most complete and solid proposal for *Database Access and Integration Services* (DAIS) on the Grid is described in [83]. The proposal is independent of any Grid implementation and of any data model or database access language. The following are the proposed database services: *Database discovery, database statements* (query, update, bulk load, schema update), *delivery system* (set source, add channel, deliver), *basic transaction interface* (start transaction, rollback, commit), *distributed transactions* (start transaction, prepare commit), *database metadata* (content description, capability description), *virtual databases* (distributed query service), *selective replication* and *management services*.

**Spitfire.** Spitfire [113, 114] provides a uniform way to access relational database management systems (RDBMSs) through standard Grid protocols and published Grid interfaces. Users can not directly issue a query in Spitfire, but Spitfire defines a set of abstract operations that allow users to interact with the database. Query results can be returned in different formats, e.g. in the Spitfire result set format or as XML. In version 1.1.0 of Spitfire architecture, GSI enabled HTTPS is used between the client and server, and JDBC is used between the server and the underlying RDBMS. Version 2.0.0 Beta of Spitfire introduces a Web Services API. It uses SOAP-RPC for the communication between the client and the server.

**GridFTP and Replica Manager.** [115] describes two services for Data Grid: data transport and replica management. **GridFTP** is proposed for data transfer, which extends the standard FTP protocol to include a superset of features offered by the various Grid storage systems currently in use. The GridFTP protocol includes features like Grid security infrastructure, third-party control of data transfer, parallel or striped data transfer and support for reliable and restartable data transfer. The **replica management component** is responsible for managing the replication of complete and partial copies of data sets, which are defined as collections of files. Replica management services include creating new copies of complete or partial collection of files, registering these copies in a Replica Catalogue, and allowing users and applications to query the catalogue to find all existing copies of a particular file or collection of files. The role of the **replica catalogue** is to map a unique logical file name to a possibly different physical name for the file on a particular storage system.

**Other Resource Management-Related Technologies**

Besides the Grid and Data Grid, a number of other technologies and projects can be named that are related to resource management. One such technology is metacom-
puting. A metacomputer is a set of heterogeneous computing resources turned into a single, uniformly accessible computing environment [116]. Another such technology is (distributed) object management. Below, three examples of these technologies are provided.

**Polder.** The Polder computing environment is a Grid-like metacomputing environment [117, 116, 118]. Polder is an experimental environment comprising software and hardware for high performance computing and interactive simulation in the field of computational science [118]. Its elements comprise the global dynamic job placement and migration, dynamic task migration, monitoring, and resource management simulation [116]. It makes use of existing computing environments and provides uniform access for their control, accounting, and monitoring. Polder predates the Grid, and adheres to a municipal area network and intranet paradigm rather than the global computing environment of Grid (which is based on the wide area network and internet paradigm). There have been some extensions to Polder to incorporate some of the Globus tools (e.g. for security and for single sign on), Dynamite for migration of tasks in a parallel program [118], and HLA for building interactive distributed simulation applications.

**High Level Architecture.** High Level Architecture (HLA) is a distributed architecture for design and execution of distributed simulation models [119, 120]. It has been designed to facilitate interoperability among simulations and promote reuse of simulations and their components. The HLA is composed of three major components [121]: *HLA rules* comprise five federation and five federate rules that describe the responsibilities of simulations with respect to the HLA Runtime Infrastructure (RTI) in an HLA compliant federation [119]. *HLA Interface Specification* describes the functional interface between simulations (federates) and the RTI, and defines how RTI services are accessed. Object models in HLA describe the set of shared objects in a simulation or federation, the attributes and interactions of these objects [119]. The *HLA Object Model Template* provides a specification of the common format and structure for documenting HLA object models for federations and simulations [121]. In an HLA application, any number of physically distributed simulation systems can be brought together into a unified simulation environment to address the needs of new applications. These types of environments are known as federations. A federation is a combination of a particular federation object model, a set of federates and the runtime infrastructure services [119]. HLA takes a multilingual approach to distributed objects. However, it is specifically targeted at distributed simulations. Currently, there are efforts to extend the HLA to use Grid services (e.g. using Grid-based data transfer protocols as an alternative to RTI communication) [122].

**ROOT.** ROOT [123] is an object oriented data analysis framework. The ROOT system provides a set of object oriented frameworks for efficiently handling and analysis of large amounts of data. In ROOT, data is defined as a set of objects, and specialized storage methods are used to get direct access to the separate attributes
of the selected objects, without having to touch the bulk of the data. Included are histogramming methods in 1, 2 and 3 dimensions, curve fitting, function evaluation, minimization, graphics and visualization classes to allow the easy setup of an analysis system that can query and process the data interactively or in batch mode. The backbone of the ROOT architecture is a layered class hierarchy with around 310 classes grouped in about 24 frameworks divided in 14 categories. In this hierarchy, most of the classes inherit from a common base class called TObject. Among the ROOT class categories, the TObject class implements common behavior for all ROOT classes; the 2D and 3D graphics classes contain both low-level and basic graphics primitives; the operating system interface handles all OS services; and the networking classes provide a way to construct client/server applications. In addition, the Runtime Type Information System maintains information about the objects (e.g. their types, methods, attributes, etc.) at run time. The ROOT system also includes a set of classes to support Input/Output from/to machine independent files, optimized for objects frequently manipulated by physicists.

3.1.4 Other Related Technologies, Paradigms and Tools

In addition to the enablers described in the previous subsections, there are other technologies and paradigms that are strongly related to the development of the necessary ICT infrastructure supporting experimentation in e-science domains. Similar to the previous enablers, these technologies and paradigms can be applied to fulfill some of the requirements. Other related technologies, paradigms and tools described here include workflow management systems, virtual organization paradigm, and toolkits for Grid software development.

Workflow Management Systems

A workflow is defined as a collection of processing steps (activities) organized to accomplish some business process [124]. An activity can be performed by one or more software systems or machines, by a person or a team, or a combination of these. In addition to activities, a workflow defines the order of activity invocations or condition(s) under which activities must be invoked (i.e. flow control) and data-flow between these activities. Activities within a workflow can themselves be workflows. A Workflow Management System is defined by the Workflow Management Coalition (WFMC) [125] as a set of tools providing support for process definition, workflow enactment, and administration and monitoring of workflow processes [126].

In short, workflows allow users to organize the activities required to accomplish a task, and specify rules for the correct execution and successful completion of the activities. Similarly, activities in an experiment follow the experiment logic, where the experiment design is a formalization of the experiment logic. When making an experiment, a scientist needs to design the experiment and execute it, by defining the activities involved, (complex) relationships among the activities, and rules and conditions for their execution. Although the design can be represented by a powerful data model, its execution requires the existence of a coordination system, such as a
workflow management system.

METUFlow2 [124], WebWork [127, 128] and WIDE [129, 130, 131] can be given among the examples of workflow management systems.

Virtual Organization Paradigm

Virtual Organizations (VOs) represent a new paradigm for collaboration among pre-existing organizations. The main motivation for collaboration is to achieve a stronger and more competitive position in the market and in adapting to the emerging market trends. Among many other available definitions of a Virtual Enterprize (VE), [132] provides a comprehensive view and defines a virtual enterprize as "a temporary alliance of enterprizes that come together to share skills or core competencies and resources in order to better respond to business opportunities, whose cooperation is supported by computer networks". The Virtual Organization (VO) concept is similar to the virtual enterprize concept, though it addresses any kind of organizations and is not limited to an alliance of for-profit enterprizes. Under the VO paradigm, a number of pre-existing organizations with some common goals come together, forming an interoperable network that acts as a single organization. In other words, VOs materialize through the selection of skills and assets from different organizations and their synthesis into a single entity [133]. A virtual organization represents a complex and dynamic entity that undergoes a sequence of stages during its life cycle. The main VO life cycle phases include the pre-VO life cycle, VO creation, VO operation/evolution, VO dissolution, and post VO life cycle [134]. Support for agility is an important requirement for the successful operation of the VO during all stages of its life cycle; namely, the VO infrastructure should provide required mechanisms and tools in order to detect and rapidly react to unpredicted environmental changes.

Application of virtual organizations concept may allow developers to fulfill many of the requirements related to collaboration, such as organizing the collaborative activities by defining and enforcing collaboration rules and conditions.

Toolkits for Grid Software Development

There are a number of toolkits providing convenient interfaces to the Grid, such as Java and Perl interfaces. These toolkits can be seen as the building blocks of Grid-based software development. Examples of such toolkits include GridPort [27] and Commodity Grid Toolkits (CoG Toolkits) [135].

3.2 Evaluation Criteria

A number of criteria have been defined for the evaluation of each system as related work. These criteria are chosen to approach the systems from different viewpoints and to evaluate their different aspects. The criteria consist of the following points:

1. identified requirements (based on Chapter 2),
2. provided functionality,
3. applied enabling technologies, standards, and paradigms (see Section 3.1),

4. employed implementation technologies.

Each system was surveyed entirely, that is, many other aspects of each system other than information management were also studied in order to capture its context properly and consistently. In this direction, although the summaries mainly emphasize the information management aspects of the studied systems, at the same time they also provide some general overviews of these systems. Similarly, the evaluation criteria are defined and the evaluations are performed with respect to the overall aspects of the systems, but with more emphasis on information management and other related issues.

Figure 3.4 provides an overview of the criteria used for the evaluation of the related systems. The organization of the tables in Appendix C, which show the evaluation results for the related systems, are based on this figure. The evaluation criteria are described in the rest of this section.

### 3.2.1 Identified Requirements Related Criteria

For each of the evaluated systems, the requirements 'explicitly' mentioned by the system are identified. Note that mentioning/identifying a specific requirement does not necessarily mean that the functionality needed to satisfy that requirement is provided. These should be seen as the requirements that motivate the development of the system under study.

The evaluation criteria here are based on the requirements described in Chapter 2. Consequently, criteria related to identified requirements are split into two: a) criteria related to the general VL requirements (infrastructure, interface, functionality, and implementation requirements); and b) criteria related to the information management requirements (modelling, storage, manipulation, collaboration, security, interoperability, and implementation requirements).

Descriptions of the evaluation criteria related to the identified general VL requirements are provided below.

**Infrastructure requirements.** Whether the system under study identifies infrastructure requirements of the scientists and their experimentations. Specifically, whether the requirements for storage, computing, networking, instrumentation facilities, or software environment have been mentioned among the infrastructure requirements.

**Interface requirements.** Whether the system identifies availability/provision of user interfaces or programming interfaces as requirements.

**Functionality requirements.** Whether the system identifies management of experiments, management of data/information handled within the system, management of available resources, or management of users as requirements. Other functionalities related to collaboration and security are also considered here.
Figure 3.4: Classification of the criteria used for the evaluation of related systems

**Implementation requirements.** Whether the system puts specific emphasis on the architecture design and on the technologies used. This could also be due to restrictions, such as the availability of a specific storage system at the developing organization.

Evaluation criteria related to the identified information management requirements are described below:

**Modelling requirements.** Whether the system identifies any information modelling
requirements. Systems could be aiming at providing support for modelling all kinds of information related to scientific experiments, either by using existing (standard) models or by developing a new data model.

**Storage requirements.** Whether the system aims at building storage facilities (e.g. archives, databases) for persisting experimental data/information. The difference from the infrastructure requirements is that, storage facility requirement as infrastructure considers hardware facilities (e.g. large capacity file servers, temporary buffers, etc.) while the focus here is on the software facilities (e.g. archiving systems, file management systems, etc.).

**Manipulation requirements.** Whether the system identifies provision of mechanisms for access and manipulation of various types of data/information handled within experiments, or aims at providing other functionalities such as query, version control or replication.

**Collaboration requirements.** Whether collaboration is one of the requirements identified. This includes sharing of data/information resources, sharing expertise, or managing collaboration activities among partners. The difference from the general collaboration functionality is that, the latter addresses sharing of all kinds of resources as well as cooperative work, while here sharing of data/information resources is addressed.

**Security requirements.** Whether the system specifically identifies issues like definition and enforcement of access rights. Security issues like authentication/authorization or single sign-on are considered as general security functionality requirements.

**Interoperability requirements.** Whether the system specifically identifies interoperability and application of standards in information modelling, sharing and exchange.

**Implementation requirements.** Whether the system puts specific emphasis on the architecture design and on the technologies used. The focus here is on the information management architecture.

### 3.2.2 Provided Functionality Related Criteria

During the study of related work, a careful and representative set of functionalities provided by the related systems was gathered. The gathered functionalities were then cross-checked against functionalities described in Section 2.2 (as part of required functionality characterization) and in Subsection 2.6.1 (as part of functionality requirements). The refined provided functionalities are categorized into six groups: experiment management, information management, resource management, user management, collaboration and security. These six groups are then detailed into a list of functionalities. Criteria related to each of these functionality groups are described below:
Experiment management functionality. This group includes all functionality related to the management of scientific experiments; such as managing experiment templates and experiment designs, executing experiments, maintaining a history of activities, and managing interpretations/conclusions of scientists (metadata). In addition, assistance provided to scientists during experimentation and session management are considered as part of the provided experiment management functionality.

Data/information management functionality. This group covers the functionality provided for management of files that are not associated with an experiment (i.e. copying, moving, deleting and transferring files), management of data sets generated by an experiment and associated to that experiment, querying of data/information which can be in files or in DBMSs, accessing data from heterogeneous data sources which are usually remotely located, integration of the data accessed from these remote sources, version control, and replication of data/information. Note that accessing data from remote sources does not imply integration of data, since integration involves understanding the semantics of the data and providing mechanisms for the semantic integration.

Resource management functionality. This group includes the functionality provided for the discovery and allocation of resources, scheduling and monitoring of jobs on these resources, and provision of descriptive information about these resources. The last criterion is whether the system supports multiple execution modes, e.g. interactive execution, batch execution.

User management functionality. This group covers the functionality for management of user accounts and roles.

Collaboration functionality. Included in this group are the functionality provided for sharing of resources (hardware, software, data/information resources), functionality to support cooperative working environments (e.g. video-conferencing, chatboxes, whiteboards), and collaboration management functionality providing advanced features such as virtual organizations for proper organization, management and controlling of collaboration activities.

Security functionality. This group covers the authentication/authorization mechanisms provided, support for single sign-on, and access rights management (i.e. definition and enforcement of access rights). No restrictions are set for the type of resources for which access rights can be defined; that is, access rights can be defined for software libraries and computational resources as well as for data sets and information.

Here, it is worth to further clarify the following three points about the criteria related to the provided functionality:

1. Evaluation is performed on a disjunctive basis. For instance, a system is considered as providing version management functionality even if it supports version
management only for one specific type of data/information among the many that it handles.

2. Management of experimental data/information is distinguished from file management. In many cases, scientific experiments/applications generate results as files, however, the evaluation criterion here is whether the system supports data/information management within the context of an experiment (experiment-based) or as individual files. Some systems relate the experimental data/information to the description of the experiment that generated the data and allow the manipulation of this data. Such systems are evaluated as providing the functionality for experimental data/information management. Other systems, however, only provide file management functionality and support management of data sets as files without any reference to the generating experiment. These systems are evaluated as providing file management functionality. Note that some systems in the former group may still provide mechanisms for file management, in which case they are also evaluated as providing file management functionality.

3. Another distinction is between the management of experimental design, management of metadata, and management of history of activities. In the literature, metadata is sometimes also used for representing experimental design and history of activities. However, in this evaluation, experimental design refers to the steps involved in an experiment and to the relationships among them, such as the ordering of steps. History of activities represents the descriptions of the actions performed on the data sets, usually with enough detail for the regeneration of the data sets. Metadata refers to both descriptions of data sets and interpretations of experiment results (i.e. scientists' conclusions). Furthermore, information about the system itself, such as information about available resources or users is also considered as (system) metadata.

### 3.2.3 Applied Enabling Technologies, Standards and Paradigms Related Criteria

In Section 3.1 the technologies, paradigms and standards that enable the development of supporting ICT infrastructures have been described. The evaluation criterion here is whether these enablers have been applied for the development of the system being evaluated.

### 3.2.4 Employed Implementation Technologies Related Criteria

Criteria in this category are aimed to evaluate the systems based on the technologies that they employ for implementation. The implementation technologies are grouped into four: Supported platforms (Unix, Windows, Linux) for server side software and client side software (where applicable), programming languages used for software development (C/C++, Java, Fortran, scripting languages), internal databases used for
data/information storage (relational, object, native XML, file-based), and other technologies (Web technology and XML).

### 3.3 Classification of Related Work

In order to achieve a systematic study and evaluation of the related work and to organize the evaluation results, first a classification is performed. This classification of related systems is carried out based on the main aspects on which these systems focus when addressing scientific experimentations. Four aspects of experimentation considered here for this classification are: context, instrumentation, computation and metadata. The *context* aspect addresses the goal of the experiment and descriptions of the methodology, techniques and mechanisms used to achieve this goal. The context can be extended to also cover descriptions of the activities and data elements involved in the experiment. The *computation* aspect addresses the computational activities in an experiment, such as running a partial differential equation solver program, a simulation program, or data processing programs. A computational activity can be a single process, or a set of processes composed together in a bigger computation unit. The *instrumentation* aspect addresses the instruments used during an experiment. This aspect covers both the descriptions of instruments and their usage. Instrumentation takes place in scientific experiments either by their manual control and usage, by their remote controlling, or in a simulation environment by virtually operating a simulated instrument. The *metadata* aspect of an experiment addresses the descriptions of the data sets that are generated by experiments.

Consequently, the studied related systems are classified into the following five groups (see also Figure 3.5):

1. **Context-oriented systems** (addressed in Subsection 3.4.1)
2. **Computation-oriented systems** (addressed in Subsection 3.4.2)
3. **Instrumentation-oriented systems** (addressed in Subsection 3.4.3)
4. **Metadata-oriented systems** (addressed in Subsection 3.4.4)
5. **Others** (addressed in Subsection 3.4.5)

In Figure 3.5, example systems considered belonging to every one of these classes are also listed. Each of these classification groups is further described below, followed by an overview of the studied systems which are classified according to these groups.

**Context-oriented systems.** The main focus of context-oriented systems in experimentation is the experiment design; in other words, the meaning of the experiment. These systems support the scientists during experiment modelling; that is, when defining the goal of the experiment, formulating the goal as an experimental procedure, defining experiment conditions and variables, and selecting the techniques, tools to use. Some systems provide the additional functionality for experiment execution and
results analysis, however, they are considered among the context-oriented systems (e.g. Zoo).

**Computation-oriented systems.** Systems in this group approach experimentation from a computational point of view, and consider experiments as *jobs*. Some of these systems allow scientists to compose jobs by connecting a number of software components to each other and execute the jobs (e.g. Gateway), while others offer a pre-defined set of software components that perform a specific task in a domain (e.g. BioPSE).

**Instrumentation-oriented systems.** Systems in this group mainly address issues related to instrumentation. These systems provide the necessary functionality for off-line or on-line operation of a real instrument, or for virtual operation of a simulated instrument. In case of off-line operation, the instrument is operated by a technician/operator, and mechanisms are provided for communication between the scientist and the operator, for instance to set the parameter values. Some systems in this group also provide cooperative work functionality, that allow scientists to join an instrumentation session, or to discuss the obtained results using diverse tools (e.g. whiteboard).

**Metadata-oriented systems.** Metadata-oriented systems mainly focus on the management of descriptions of the data sets generated by experiments. These systems provide a metadata catalogue containing, for instance, information about the owner, information about the physical location of the data sets, or in some cases information about the contents of the data sets. The commonly provided functionality is querying the metadata catalogue. Some systems (e.g. SRB - Storage
Resource Broker) additionally provide mechanisms for the management of the data sets themselves.

Others. This group contains the systems that can not be classified in any of the previous groups, either because they are too restrictive to be included in any of the groups, or because they are too extensive to be classified in a single group. The only system in this group is the "Enter The Grid" portal, which provides a Web-based entry point to a number of Grid-related resources. This system is classified in this group because it is too restrictive. Note that later on in Section 3.6, the VLAM-G/VIMCO will be addressed as another system to be studied under this category due to the fact that its functionalities are too extensive to be classified in a single group. Therefore, Figure 3.5 classifies VLAM-G/VIMCO in this category in addition to the Enter The Grid portal.

This chapter contains the summaries of 28 systems that were studied and evaluated. Names of the studied systems, and their classification into the five groups can be found in Figure 3.5.

Although the related work can not be limited to only the list given in Figure 3.5, it contains the most representative and significant examples of different types of support environments considered in this dissertation for scientific experimentations.

Another survey of related work can be found in [36]. This survey focuses only on a restricted set of systems, called as Grid Computing Environments (GCEs), and classifies them as technology for building GCE systems, largely problem solving environments, and largely basic GCE shell portals.

3.4 Overview of the Studied Systems

In this section, brief information about each system studied as part of the related work is provided. The summaries are presented based on the classification described in Section 3.3. Each summary provides a general overview of a system with emphasis on the management of data/information. Evaluation of the systems with respect to the criteria is provided in the following section.

3.4.1 Context-Oriented Systems

This subsection includes examples of context-oriented systems, which put more emphasis on the contexts of experiments. In specific, systems presented here address management of experiment templates, designs, history of activities, data/information, and metadata. Systems summarized here are Zoo, PELLPACK, SoftLab, Sieve/Symphony and OPM.

Zoo. Zoo [136] is a Desktop Experiment Management Environment. A key objective set for the development of Zoo is to achieve an integrated software package with uniform interfaces that supports the entire life-cycle of an experimental study.
3.4. Overview of the Studied Systems

Zoo is developed as a generic system, which can be customized to support specific experiments.

The Zoo architecture is based on an extensive use of conceptual schemas. In Zoo, schemas are called to play new roles other than capturing the structure and constraints of data stored in a database, namely: a) the schema becomes the formal document describing the experiment, and b) the schema serves as the template for specifying data and experiments [137]. Scientists can define schemas without thinking as expressing database structures. A schema does not necessarily have to be associated with a database, thus it may exist uninstantiated.

The modular architecture of Zoo comprises several modules: Moose [137, 136] is the object-oriented data model of Zoo. The various kinds of classes in Moose include primitive classes, tuple classes, and collection classes. Any relationship from one class to another may be specified as derived. In a derived relationship, successor of the relationship is constructed based on other objects that are related to the predecessor of the relationship. The construction may be through a query or may require processing by an external system that receives as input a file containing these other objects. Similarly, inheritance is also represented as derived relationships. Opossum is a schema manager and Squid is a query/update manager, both of which make up the Zoo user interface [136]. They are generic visual systems whose inputs are files with mappings between the data model or query and the visual model. Upon receiving these mapping files, Opossum or Squid is customized to operate for the specific data model and visualization style, thus, schemas and queries may be visualized in different ways. Emu is responsible for transforming user requests into actions at external systems. It interacts both with Horse to retrieve the necessary information related to the user request, and with agents. Agents are the intermediary components between Zoo and external systems (similar to wrappers/mediators). Turtle is the translator from Moose objects to Ascii files and vice versa, using a map file that contains specifications for how various parts of complex object correspond to parts of an external file. Frog is a visual tool for users to generate these map files. Horse is the backbone of the Zoo environment. It is implemented in a layered mode. At the back-end, it uses Informix RDBMS as the storage system. At the front-end, it accepts Moose data definition and Fox query and update requests. All Zoo modules communicate each other with a messaging system, passing data in the shipping form. Shipping form is a serialized form of objects into structured text files.

All information treated in Zoo is stored in different databases, including databases for experimental studies, interesting results of queries as well the queries themselves, visual representations of database schemas, and models and map files. Schemas for these data are represented recursively, that is, the schema of each one of the above databases is represented as objects in higher-level meta-database, until some root database is reached. The three root databases in Zoo include one database for storing all user defined schemas, one for storing queries, and one for storing all visual models used for visualization. For example, the meta-schema for user-defined Moose schemas contains classes named Schema, ClassSet, Class, RelationshipSet, Relationship, DerivationRuleSet, and DerivationRule. User defined Moose schemas are represented as objects in this meta-schema. Other data is treated similarly.
In Zoo, experimentation and any other form of external processing is achieved through the derived relationship mechanism. For instance, users define an experiment in a Moose schema by specifying the experiment, its input and output elements. Here, output elements are described as derived relationships based on the input elements. When the user wants to retrieve data about an experiment, Horse returns the data from the storage if it already exists. Otherwise, it initiates the data generation (i.e. experiment execution) through Emu, which performs the derivation rules specified for this experiment, for instance, by calling external systems through agents. Then the results are returned to the user. For experiment management, Task and Agent system classes are defined in Zoo. At any point, a Task object exists in Zoo corresponding to an external computation/activity, with references to the status, output, and input objects. There is one Agent object for each registered Agent. For each request to perform an experiment, several Task objects are created by Horse corresponding to each external activity. For the maintenance of status information, a shadow subclass of the class representing the external derivation process is created, which contains the run-time objects representing the derivation.

PELLPACK. PELLPACK [46, 138] is a problem solving environment (PSE) for solving certain classes of partial differential equations (PDEs) on sequential and multicomputer platforms. PELLPACK provides an interactive graphical user interface for specifying the PDE model, its solution method and post-processing, supported by a symbolic system and libraries of sequential and parallel solvers. Users can create experimental designs either directly using the PELLPACK language or using the GUI. The design consists of the problem specification, solution method, parameters, conditions, etc. The designs in the PELLPACK language can be saved and loaded later on. During the PDE framework specification, a symbolic processing and code generation is performed on the problem specification of the user in the format required by the selected solver. The result is a PELLPACK template for the problem to be solved by the selected solver. Users then enter the crucial pieces of information that define the problem parameters.

The PELLPACK architecture can be viewed in terms of its software layers: The GUI serves for PELLPACK program building, and solution and performance visualization/analysis. GUI supports multiple sessions each representing a single problem to be solved. The tools available to users depend on the type of the session (e.g. 1-D, 2-D, etc.). Different filters are applied to display the options for the next step as the user continues with the specification of the problem and methods. There are editors with associated tools. For instance, the session editor reflects the current status by displaying the specification in the PELLPACK language, while graphical and interactive toolbox editors are used to create/modify the PDE objects. The very high-level language interface of PELLPACK architecture allows users to specify PDE problems and solution methods using a high-level PDE-specific language. In PELLPACK, a PDE problem is defined in terms of the PDE objects involved: PDE equations, domain of definition, boundary and initial conditions, solution strategy, output requirements and option parameters. Textual representation of the PDE objects and its syntax comprise the PELLPACK language, which underlies all components. Users can write
a program in this language directly or use the GUI to automatically generate the PDE program. A language processor translates the control program into a FORTRAN control program that invokes the appropriate library components to solve the problem according to the user’s specifications. In order to generate the top-level calls to the library modules in the control program, the language processor uses a module database to obtain necessary information such as memory requirements. The Procedural Language (FORTRAN) Interface of PELLPACK is based on a decomposition of the PDE problem and the solution methods into their constituent parts, such as PDE equations, domain, boundary conditions, etc. At run time, each problem part is represented using a set of standard data structures and/or functions. The control program in FORTRAN, which uses the solver libraries and these data structures to solve the program, is then compiled for a specified host, and linked with the necessary library modules to generate the executable binaries. Depending on whether parallel algorithms are used, there can be one or more programs, and one or more output solutions. In case of a parallel algorithm, the generated local solutions are collected by the system to generate the global solution. Finally, the PELLPACK infrastructure consists of a number of solver libraries that comply with the component interfaces. Libraries that do not comply are represented as the system infrastructure.

**SoftLab.** The SoftLab project [62] addresses the issues involved in the design and implementation of a virtual laboratory that simulates and controls the functionality of a wet/dry prototyping laboratory. All instruments used during the experimental process have visual representations in SoftLab called virtual instruments, which look like and operate just as their physical counterparts. SoftLab provides four main functionalities that are presented to users as scenarios. During the *physical experimentation scenario*, scientists remotely control laboratory instruments and monitor the experiment via animation of the virtual instruments. Users can control data collection and extraction during the experiment, and visualize the results when the experiment is completed. Experiments can be stored in an experiment database, saving all information necessary to re-create the experiment. During the *virtual experimentation scenario*, scientists set up the virtual instruments and experimental process, and perform data collection, visualization, and saving of experiments exactly in the same manner as physical experiments, only the experiment is actually a simulation. *Experiment playback scenario* is used to retrieve physical and simulated experiments from the database, and play back for review, comparison, or data visualization. *Training scenario* uses multimedia annotations for the training of students.

The software architecture consists of three main functional components: *Kernel* providing visualization, database access, graphical representation and animation of virtual instruments, physical instrument control, simulator input interface, and output transformation; *four scenario components* allowing parameter setting, running physical or simulation experiments, retrieving, analyzing and visualizing data, and saving experiments; and *experiment and parameter databases*. Specific software, such as the PELLPACK [46] problem solvers can be attached to the system and used within SoftLab.
Sieve and Symphony. Research work described in [48] presents two tools called Sieve and Symphony, with the key themes defined as ‘ability to do compositional modelling, collaboration, and exploiting the larger context of the application’. The last theme includes previously performed experiments and obtained results in addition to the current experiment and data sets. The high-level architecture defined in this work consists of six layers; from top to bottom: user interface (PSEs, portals), model definition, parameter definition, simulation definitions, Grid services, and computational Grid. The bottom two layers represent the low-level Grid services and the resources available through these services. Model is defined as a directed graph of specific executable pieces defining the control and data flow in a computation. Model instance is a model with all parameters specified. A simulation, on the other hand, is a model instance assigned to and run on a particular computational resource on the Grid. Emphasis in [48] is given to XML-based semi-structured representations for models. Here, XML is not seen as a data format, rather a binding from representations to models, specified using the binding schema markup language (BSML). BSML describes the binding of the data elements specified in an XML document to the functions in the target; for instance, passing the structures defined in XML as parameters to MATLAB by converting the whole XML document into a MATLAB script. By storing the data and available binding types in a database, the same model can be converted into different representations required by different environments. Previously executed simulations are stored in a database along with the data generated by them and the appropriate links between the data and the simulation descriptions. This database of simulations and data can be queried to obtain the performance history of the previous simulations, thus to choose the model to use and the parameters to instantiate the model as well as to determine the computational resources to schedule and run the simulation.

Sieve [48] is a composition environment to define data or control flows by connecting components to each other. Components are implemented as JavaBeans, representing individual codes, optimization tools, visualization tools, etc. Users define the actions performed by the connections and the data format flowing through these connections. Data compatibility is checked during design time, if a conflict occurs, a suitable converter is searched and inserted between components. Users can enter a collaborative session by specifying at the start which session to join. In a collaborative session, all users can view the same visualization and the same annotations to the model. Symphony [48] is a component based framework for composing, saving, sharing and executing simulations. Simulations in Symphony are also represented as directed acyclic graphs. These simulations can be saved for later usage or shared with other users. The developed back-end execution environment is RMI based, and provides access to resources local to the machine on which the RMI server running. The defined simulation is represented as JavaBeans in the front-end environment, which have corresponding processes in the back-end. The front-end beans coordinate the correct execution of the back-end processes.

OPM. The Object Protocol Model (OPM) [139, 140] is a data model that allows specifying database structures and queries in terms of objects and protocols. It pro-
vides a protocol class for modelling experiments, which can be associated with input and output attributes. Protocols can consist of sub-protocols, or can be connected to other protocols via alternative, optional or sequential paths (protocol steps). Update rules are defined for expressing the effect of updates on an element on other related elements. Attributes in an OPM class can be simple or tuple attributes. Derived attributes are simple attributes that have values derived from the values of other attributes using either arithmetic expressions, aggregate functions, or attribute composition. A derived object, on the other hand, can be a subclass, a superclass, or an aggregate class. In OPM, protocol modelling is characterized by the recursive specification (expansion) of generic protocols in terms of alternative sub-protocols, sequences of sub-protocols, and optional sub-protocols. In addition to regular attributes, protocol classes have connection and special input/output attributes. Connection attributes are used to reference the superclass, subclass, predecessor, and successor protocols of the defining protocol class. Input and output attributes are used to represent the input and output data regarding the experiment modelled by the protocol class.

The data management tools of OPM are used to define database schemas and to retrieve and store protocol information from the underlying relational database [140]. The OPM data management tools consist of the Schema Editor, Schema Translator, Browsing and Query Tool, and relational to OPM Schema Conversion Tool. OPM schemas are kept in ASCII files. OPM schemas can be specified using either a regular text editor or using the graphical OPM Schema Editor. OPM Schema Editor allows specifying interactively and incrementally object and protocol structures. The OPM Schema Translator translates OPM schemas into relational database definitions and database procedures implementing the OPM retrieval and update methods (of the OPM classes). OPM Browsing and Querying Tool allows specifying OPM queries and browsing the results of these queries. The relational to OPM Schema Conversion Tool can be applied to existing relational database definitions, to generate the data structure in OPM data model.

3.4.2 Computation-Oriented Systems

The systems summarized in this subsection put the emphasis on the computational aspects of experimentation. These systems consider experiments as jobs to be executed on a distributed, high-performance computing infrastructure. A job may be either composed of executable components, or can be a single executable. In the former case, users compose the jobs from executables and define input/output and parameter values for each executable, while in the latter case users only select input/output and set the parameter values. Examples of computation-oriented systems included here are Gateway, UNICORE, SCIRun (together with BioPSE and Uintah), Cactus, ASC, Ecce, The Virtual Laboratory, and FAST (together with RemoteFAST and FASTExpeditions).

Gateway Computational Toolkit. The primary focus of the Gateway Computational Toolkit [141, 30, 28, 142] is to assist PC-based researchers in the use
of high-performance computing (HPC) resources. XML descriptors for describing applications, HPC resources and services are stored at the Web server to generate dynamic content for the user and back-end requests. Gateway implements the following services: Secure identification and authorization, information services for accessing resource descriptors, batch script generation, job submission and monitoring, file transfer, remote file access and manipulation, session archiving and editing.

Gateway employs a multi-tier architecture. The Web-based user interface provides three usage tracks [30]: Code selection allows users to start a new problem, make an initial request for resources, and submit the job. The problem archive allows users to revisit and edit old problem sessions, and to resubmit them as new sessions. Administration track allows privileged users to add applications and host computers to the portal, and modify the properties of these entities. When the user is authenticated and authorized, a session is created with user, problem, and session contexts. User context is the base, which contains problem and session contexts. Users can compose their applications using a set of modules in the front-end, which are then instantiated at different nodes. The middle tier of Gateway architecture contains a Web server maintaining a number of Java components, which are implemented as JavaBeans and communicate with each other through CORBA. They either handle specific server-side tasks or act as a proxy for a remote object. There is one Gateway server on each node. In case of remote object invocations, the remote object is created and maintained by the Gateway server running on the remote host, and a proxy is created locally to access that object. Resources in the back-end tier are accessed using remote shell operations.

UNICORE. UNICORE [31, 143] aims at providing a uniform interface for job preparation and control. It supports only batch jobs. UNICORE defines a protocol for job descriptions called Abstract Job Object (AJO). An AJO can recursively contain other AJOs and/or atomic tasks, together with their dependencies, information about the destination site, user, and site-specific security. The results of abstract tasks are accessed by the Job Monitor Controller and presented to the user. Users can view the jobs that were executed earlier and resubmit them. Information about resources at a UNICORE site, such as information about the system architecture, performance, operating system, and available applications and system software are kept in a resource page.

The UNICORE architecture follows a three-tier model. User level is an applet with a Job Preparation Agent and a Job Monitor Controller. UNICORE server level consists of a Web server, Java applets, resource information about the available resources at the UNICORE site, authentication system, gateway to map the user's certificate to user's id at the target system, and Network Job Supervisor which translates the job descriptions into jobs for destination systems, executes and monitors them. The batch subsystem level contains the destination systems with their batch systems and data storage. Multiple UNICORE sites can interact through their gateways. A UNICORE site in UNICORE environment is defined as a computer center offering a UNICORE server and execution hosts grouped in virtual sites. A virtual site consists of systems at one UNICORE site sharing the same data space. The file systems available at
virtual sites of a UNICORE site are named as XSpace's. An Xspace also includes the file space of the user's workstation. All data available to a UNICORE job constitute the UNICORE file space (USpace). Thus, the UNICORE data model is divided into internal data space (USpace) and external data space (XSpace). All data required during the execution of a job need to be imported into the USpace, and the output must be exported to the XSpace. The Network Job Supervisor (NJS) is responsible to transform the AJO into batch descriptions for the destination system, submit these batch jobs, create a UNICORE job directory to contain the data created before and during the job run, and collect the output and error files belonging to one UNICORE job and make them available to the user. The NJS also initiates all necessary data transfers, imports, and exports.

**SCIRun, BioPSE and Uintah.** Three different PSEs are described in [49]: SCIRun, BioPSE and Uintah.

**SCIRun** is a PSE that allows interactive construction, debugging and steering of large scale scientific computations. SCIRun provides a component model based on dataflow programming, which allows different computational and visualization components (modules) to be connected. Users define the dataflow using the available set of modules in a graphical environment, and set the necessary parameters required by each module in the dataflow. Application steering is possible either by directly changing the internal states of the modules in the dataflow, by cancelling the current operation and starting with a new one using the new parameter values, or by feedback loops in the dataflow program. SCIRun is implemented using mainly C++. Tcl/Tk is used for the user interfaces, while other components use C and Fortran. SCIRun is currently running in a standalone mode, employing shared memory architecture for the execution of the modules in the dataflow. Currently Grid tools are being utilized to move towards a more distributed programming model.

**BioPSE** addresses bioelectric field problems. It uses and extends the SCIRun core. The two major improvements introduced in BioPSE are *bridging* and *fields* classes. *Bridging* refers to mechanisms for importing and exporting data and services through re-writing existing code, making calls to third party libraries, and exchanging data through files, sockets or databases. The second improvement is the templatized *fields* classes. Fields are used to represent geometric domains and data values over that domain. The mechanism is flexible, in which the calling code do not know what type of fields will be passed to it as input. In order to provide optimization, as soon as the module identifies the type of fields, it dynamically compiles the corresponding optimization code for that fields and loads it for use during execution.

**Uintah** is another PSE developed on top of SCIRun. It focuses on providing tools for numerical simulation of accidental fires and explosions. Uintah has added the DOE interchangeable component programming model to SCIRun and the support for running under a mixed shared memory/message passing model.

**Cactus.** Cactus [144, 43, 42] is a PSE which enables parallel computation across different architectures and collaborative code development between different groups.
Cactus mainly targets simulation applications in the numerical relativity and astrophysics domains.

Cactus architecture consists of modules (called thorns) which plug into a central core code (called the flesh) which contains the APIs and infrastructure to glue the thorns together [144]. The actual functionality is in thorns. Depending on the application requirements, a scientist gathers her/his own collection of thorns together, and compiles them using a set of parameters. The collection of thorns, parameter values, platform to run, and the compilation options altogether form a configuration. Several configurations can be created from the same set of thorns. Thorns are grouped together into arrangements, depending on their functionality and applicability. For instance, the Cactus Computational Toolkit [43] includes parallel interpolator thorns, basic parallel elliptic solver thorns, thorn implementing I/O routines using different libraries and thorns providing transparent parallelization to other thorns.

Data management facilities provided by Cactus are so far limited to some parallel I/O layers [144]. Tools for Grid applications are being developed within the Cactus community, including mechanisms for Grid enabled communication and I/O which target at remote file access, remote online data streaming and visualization, and remote monitoring and steering [42].

ASC. The Astrophysics Simulation Collaboratory (ASC) project [18, 45] aims to provide a Web-based problem solving framework for the astrophysics community that links users with remote Grid resources. Collaborating researchers in the field of numerical relativity form a community called the ASC Virtual Organization (ASC VO). The ASC enables users to work with the Cactus toolkit in a location independent manner using supercomputing resources that are made available to the ASC VO. Users construct simulation code with Cactus and other components located at multiple sites, compile on any computer, run on any available computer, and share the results with colleagues in a controlled fashion. Functionality provided by the ASC include among others single sign-on, user and task management, access control for administrative operations, and detection of what services are available on remote resources.

The ASC is comprised of several technologies arranged in an n-tier architecture: *Web and java clients, application servers* (Web server and a number of application services), *resource brokers* (LDAP directory service, RDBMS), and *remote resources* (LDAP directories, relational data). The relational MySQL database management system deployed in the ASC is used to store information about users and remote resources, as well as to record user activities by application servers. The database is also used to notify users by email when their tasks complete. The ASC relies upon Grid technologies for resource sharing, information services, and security. Grid services are accessed through the Java CoG Toolkit [135].

Ecce. Ecce [47, 145] is one component of an integrated molecular science software suit providing advanced computational chemistry techniques for complex chemical systems in high performance, parallel computing systems. Ecce was originally designed at object level integration. It is based on an object-oriented chemistry data
model that supports management and manipulation of computational and experimental data, and metadata. Data management component of Ecce uses an ODBMS and a common object model. Molecules, basis sets, calculations, jobs, and experiments are represented as objects in this model. Experiments are composed of tasks. Each task has a molecule subject, and consumes/produces files. Tasks are defined by their run types, theories, applications, and jobs.

Currently, this architecture is being extended to provide improved/extended functionalities and to incorporate Grid services, with a focus on data and metadata management, information services, and job submission [145]. The aim is to achieve direct access to raw data, self-describing data and relationships, schema independent data store, and separation of application level objects from the data storage. For this purpose, coupling in Ecce is reduced by moving away from a PSE-wide schema towards a services-oriented architecture. The new architecture of Ecce consists of a desktop component supporting data and resources browsing, job launching components, data monitoring and store components, data/metadata server, information service (for accessing directories, RDBMSs, files), job script service, scheduler services, and computational codes. The Distributed Authoring and Versioning (WebDAV) [78, 79] is used for the implementation. Clients willing to access data (in files) and metadata (in a hashtable-based database) use an Apache Web server extended with a DAV module. Other protocol libraries allow, for instance, using Grid resources.

Job management uses Globus and Silver MetaScheduler services. Input and output data are staged to remote nodes using ssh, rsh, or telnet. Job monitoring is achieved through a filtering job, which resides at the same node with the job, and continuously checks the output generated by the job. After filtering, the data is sent back to the user. Job script templates are stored and maintained by the information service. The script service locates a script template for a user request, queries the information service for application and machine information, substitutes these values in the script template and returns the script to the client application. Experimental designs are also stored in data/metadata store. Data and metadata management is done through the data/metadata server implemented using DAV. Experimental data is stored in binary or structured files. Metadata can be name-value pairs, where value is either simple text, or a complex object in XML structure.

The Virtual Laboratory. The Virtual Laboratory [146, 65] offers a toolset to enable distributed molecular modelling for drug design using geographically distributed resources. Drug design using molecular modelling techniques involves screening a large number of molecules in a chemical database (CDB) to identify those that are potential drugs. The Virtual Laboratory tools transform an existing application into a parameter sweep application for executing jobs that dock molecules in a CDB in parallel on distributed resources without making any changes to the application.

The Virtual Laboratory builds on the existing Grid technologies and tools. Its architecture consists of the following components: The Nimrod-G tools for creating parameter sweep applications provide a simple declarative language and GUI tools for parameterization of application input data files and creation of task-scripts to be executed by each job. The Nimrod-G Grid Resource Broker identifies the user and ap-
application processing requirements and selects a Grid resource combination accordingly. 

*CDB* consists of an ASCII file in a specific format containing molecule information. An indexing tool creates an index with entries consisting of the molecule id and the size of the molecule in the CDB file. Replicas of the CDB are created, and at runtime the most suitable CDB is selected. The CDB services provided are connect, get molecule, and disconnect.

A number of files are involved when composing a parameter sweep application: A *configuration input file* specifies as name-value pairs the parameters to be used during application execution. In a *parameterized input file* values of parameters are replaced by substitution markers, which are substituted with real values at run-time. A *plan file* consists of the parameters part which defines parameter types and their valid value sets using substitution markers, and the task specification part defines a series of operations that each job needs to perform to dock a molecule before and after the application execution. Once the plan file is created, it is submitted to a job generation tool, which creates a run file that contains a job for each combination of parameters. Then the Nimrod-G schedules each docking job to a Grid resource, by first submitting an agent to the remote node. The agent, after starting execution, contacts the Nimrod-G dispatcher for job task information, which contains the instructions for executing the job. It copies the necessary input files (specified in the plan file), performs parameter substitution, executes programs (also specified in the plan file), and ships results back to Nimrod-G user. During the job scheduling, users can select the optimization strategy to finish the jobs within the deadline and/or the budget.

**FAST, RemoteFAST and FASTExpeditions.** The work described in [147] approaches PSEs from the collaboration perspective, specifically from the point of view of collaborative analysis tools supporting high performance graphics. The desired features for these tools include the following: a user interface with highly interactive, high resolution, dynamic, 3D graphics; possibility for all scientists to see the same view of the analysis simultaneously; transfer of control between users; efficient usage of commonly available network bandwidths; good responsiveness; and ability to record, edit, and replay segments of an analysis session in asynchronous mode. The proposed approach is to first distribute the data to be analyzed to the remote sites, and then only send the events for controlling the applications that run at all sites.

**FAST** [148] is a visual analysis tool for physics simulations, which runs on workstations and launches parallel tasks that are controlled by a central hub. Event handlers in each parallel task reports events to the controlling central hub as an ASCII script, which records these events in a journal file. The hub then returns the script back to the parallel task for execution. Playback of an analysis session is achieved by having the hub read the ASCII script corresponding to the session from a journal file and send it to the appropriate parallel tasks. Journal files are condensed to reduce the large number of mouse transformations into a smaller number of but equivalent transformations. **RemoteFAST** [149] is the tool enabling synchronous collaborative visualization. To start a synchronous session, it first distributes the data files to each site and launches FAST at each site. It then launches a daemon at each site dedi-
cated to efficient passing of events between the sites. During a session, the controlling RemoteFAST detects the command scripts as they are being recorded into journal files, and send them to the other sites. Controlled sites receives these scripts as being retrieved from the local disk and passes them to the RemoteFAST hub, which then forwards them to FAST for execution. For asynchronous collaborative visualization, FASTexpeditions [149] tool is created. The data to be analyzed and journal files are stored in a Web server. Upon request, the data and the command scripts are sent to the user site for execution. A utility tool enables the saving of local journal files as Web pages.

3.4.3 Instrumentation-Oriented Systems

Systems summarized here constitute examples of instrumentation-oriented systems. The common denominator of these systems is the existence of a real or simulated instrument, which is operated from within the system either on-line (remotely controlled) or off-line (controlled by an operator). Instrumentation-oriented systems presented in this subsection are Virtual NMR Facility, Virtual Lab, Virtual Geotechnical Lab, Remote Lab, Tele-Actor, and VCLab.

Virtual NMR Facility. The Virtual NMR Facility (VNMRF) [63] is developed to support researchers in using high field nuclear magnetic resonance spectrometers. The VNMRF allows remote researchers (who are awarded time) to run instruments, see laboratories, talk with their colleagues, work with them in analysis programs, and share notebooks via the Internet.

In the VNMRF, programming interfaces of existing tools for real-time and asynchronous collaboration were used to provide custom research environments tailored to NMR, such as Collaborative Research Environment (CORE2000) and Electronic Laboratory Notebook (ELN). CORE2000 is a suite of real-time collaboration tools. It contains shared computer screens, remote cameras, third party audio and videoconferencing tools, whiteboard, chatbox, and other tools. When a user starts or joins a session, a palette of icons is displayed representing the available tools. CORE2000 also has a programming interface. The ELN is designed to allow distributed teams to record and share a wide range of notes, sketches, graphs, pictures, and other information. The ELN includes editors to create text, equations and whiteboard sketches, to capture screen images, and to upload arbitrary files. Secure instrument control and data access in VNMRF is achieved through secure shells along with encryption for X-windows. Only one remote user can control the spectrometer at a time, while others can observe through capturing and sharing the device console. Users operate the spectrometer in coordination with a human operator called consultant. Consultants grant access to instruments, receive and insert samples, provide guidance on spectrometer operation, then monitor the use of the instrument. With the help of above mentioned tools and the consultant, users can remotely control the instrument, retrieve the data out of the instrument and share it with colleagues who are observing the instrument operation in the joint session, discuss the results, and record the discussion results along with the actual measurements.
Virtual Lab. The remote Virtual Lab [150, 67] is used for teaching control theory. Experiments in Virtual Lab use simple controllers with fixed structure. Students select the controller algorithm and define controller parameters remotely. To perform any experiment, students submit a request. After downloading the instructions for the experiment, students may be required to perform some advanced preparations, and they can continue only if their preparations are approved by the responsible professor. Then the student logs in to the system and books the experimentation time. The booking time is stored in the database, and during the experiment time, it is used to ensure exclusive access. The database contains three tables for account, schedule and log information. The laboratory administrator interface is used for creating and deleting accounts, defining time quotas for each experiment, defining time slots for booking, and analyzing log messages. The experimentation interface consists of a video/audio stream applet and a control applet. After login, the server checks whether there is enough time for the experiment. Client side sends requests as data telegrams to the server, which acknowledges with a status message. Dialog windows are used for the system identification and controller design purposes. The resulting data is stored in a flat file, which can be read and displayed by the analyzer applet.

Virtual Geotechnical Lab. The Virtual Geotechnical Lab [151] is an example of virtual laboratories for civil engineering education. Users of the Virtual Geotechnical Lab can assess their prerequisite knowledge by answering a set of questions, where each question is graded and a record is kept about each user’s progress. Then the user begins the lab test by preparing a specimen for the test through a guided interactive exercise. The virtual apparatus is displayed to the user, who then sets up the test specimen in the apparatus with guidance and begins the test by adding loads and switching on any motor that is part of the virtual apparatus. The results are then displayed on simulated panel meters. When the test is completed, the user is guided to perform interactive tasks and to do any calculations necessary for the interpretation of the results. Finally, the user is presented with a practical scenario to which s/he can apply her/his results.

Remote Lab. In [56] and [152], a remote laboratory approach for experimentation with real plants using the WWW is presented. It is used to remotely operate the Laboratory-Scale Optical Tracker. Its architecture consists of the following components: Client is a browser that runs on the student’s computer to access and operate the remote experiment. It runs the real-time code corresponding to the implemented SIMULINK model. The general control panel allows the student to start/stop experiment sessions and to reset an experiment with specific start conditions. User console panel allows to set parameters, and execute experiment-specific commands. Communication server runs MATLAB/SIMULINK combined with the WinCon server to communicate with the real-time client. Interactive plant user interface is implemented using VRML, and enables the user to operate the real plant manually by mouse actions. The graphical representation of measured signals (generated by MATLAB) is
displayed in the real-time signal graphic display. Real-time video and audio panes allow the users to inspect the real lab through a camera and a microphone.

**Tele-Actor.** The Tele-Actor [153, 66] is a collaborative online tele-operation system for distance learning that allows many students to simultaneously share the control of a single mobile resource. The mobile resource can be a mobile robot or a human tele-actor, with wireless video and audio connections. Based on the video and audio received from the tele-actor, users can vote on the next action to be performed by the tele-actor. A middle-tier server receives the votes, and applies the algorithm to select where to go based on the spatial and temporal properties of the votes sent by users.

**VCLab.** The Virtual Control Lab (VCLab) [154, 152] replaces a control laboratory with a simulated one. It allows users to walk around in the building from one lab to another and perform some virtual (simulated) operations. It ties together the computational and simulation machines of MATLAB/SIMULINK to Web components running in a browser. In order to integrate MATLAB functions into the Web page, several browser extensions are plugged in, such as MATLAB plug-in to link the computational engine of MATLAB with the browser, or the VRML plug-in to animate 3D models of dynamical plants.

### 3.4.4 Metadata-Oriented Systems

Metadata-oriented systems focus on maintaining a metadata catalogue for the data sets generated by experiments, thereby allowing scientists to locate data sets based on attribute-value searches. Although the types of the managed metadata differs from system to system, common metadata include information about the owner of the data set, hierarchical structure of the data set (files, collections, and sub-collections included in the data set), and information about the physical location of the data set. Systems summarized in this subsection are CLRC Data Portal, SRB, GriPhyN and Chimera, MCS, MDMS, and Virtual Sky.

**CLRC Data Portal.** Aim of the CLRC Data Portal [29] is to offer a single point for reviewing and searching the contents of all the CLRC data sources through the use of a central catalogue holding metadata about all these resources. A prototype based on these ideas is being developed. The CLRC metadata model attempts to capture scientific activities at different levels. Generically, all activities are called *studies*. Each study has an investigator and study information describing the details. *Programs* are studies that have a common theme and a common source of funding. Programs can be single projects or a series of projects. Every program can be associated with a series of sub-investigations. *Investigations* are studies that have direct links to data holdings. Experiments, measurements and simulations are specific types of investigations. *Data holdings* have three layers: investigation, logical data, and physical files. One investigation generates a sequence of logical data sets (e.g. raw,
intermediate, or final data sets), and each data set is instantiated via a set of physical files. There may be different versions of data sets in a data holding.

The system has three main components: A web-based user interface which is implemented using a standard Web browser and communicates with the metadata catalogue in XML, a metadata catalogue, and data resource interfaces which are implemented as wrappers around existing data sources.

**SRB.** One of the key requirements for the development of the Storage Resource Broker (SRB) [155, 156, 157, 158] is to support data intensive computing, which involves both providing high performance I/O to massive data and providing digital library capabilities for storage, search, and retrieval of scientific data. SRB is developed to provide seamless access to data stored on a variety of storage resources, including filesystems (UNIX), database systems (Oracle, DB2, Illustra), and archival storage systems (HPSS, Unitree). It implements two interfaces for these storage systems: One is a UNIX-like I/O interface, the other is an interface that supports get and put operations.

Data is organized as a hierarchy of collections and sub-collections. A collection (recursively) contains data items and/or sub-collections. SRB distinguishes the physical and logical storage resources (PSRs and LSRs, respectively). A set of PSRs can be combined into a single LSR. Access control is supported at the collection, sub-collection, or data item level. Users can issue tickets to other users to grant them access to certain data items for a certain time period or for a certain number of times. SRB employs a metadata catalogue service. The catalogue contains descriptive and system metadata about the data objects stored by SRB. SRB metadata includes location information for PSRs and for data items; metadata that is used for implementing access control, collection/sub-collection hierarchy, and ticket mechanism; and descriptions of the contents of collections/data items (as attribute-based properties). SRB provides a set of APIs for querying and updating the catalogue, through which attribute-based access to data collections and items is supported. The catalogue is implemented using both Oracle and DB2.

Several SRB servers can be linked to each other, each managing a set of storage resources. If a request arrives to a server for a data item which is not managed by that server, then the server initiates a request to the server holding the data item. An SRB server consists of one or more SRB Masters and a number of SRB Agents. An SRB Master receives requests from clients and dispatches SRB Agents to handle the request. SRB Agents use the metadata catalogue to obtain necessary system metadata. Clients can perform proxy operations or third party operations, that is, for instance moving a file from one storage to another without the involvement of the client itself. The client APIs of SRB include APIs for query/update of metadata, connecting to the server, and for creation/manipulation of data items. MySRB [157] is a Web based interface to SRB that allows data movement operations and operations on user defined and standardized metadata (e.g. Dublin Core).

**Chimera.** The driving force for the GriPhyN (Grid Physics Network) [159] project is the need for geographically dispersed extraction of complex scientific information
from very large collections of measured data. GriPhyN focuses on the concept of Virtual Data, which encompasses a virtual space of data products derived from experimental data. GriPhyN primarily focuses on the development of a Virtual Data Toolkit (VDT). VDT contains two types of software, namely fundamental Grid software (e.g., Condor, GDMP, Globus), and virtual data software developed by the GriPhyN project to work with Virtual Data.

The Chimeria Virtual Data System [160] is being developed as part of the VDT in GriPhyN. The main argument of Chimeria is as follows: “Much scientific data is not obtained from measurements but rather derived from other data by the application of computational procedures. Explicit representation of these procedures can enable documentation of data provenance, discovery of available methods, and on-demand data generation (called as virtual data)”.

In other words, the aim is to track how data products are derived with sufficient precision, so that one can recreate data products from this knowledge. The Chimeria architecture comprises a Virtual Data Catalogue (VDC) for representing data derivation procedures and derived data, and a Virtual Data Language Interpreter that translates user requests to data definition and query operations on the database. The VDC schema defines a set of relations used to capture and formalize descriptions of how a program can be invoked, and to record its invocations. The three main entities in the schema are transformation (an executable program), derivation (an execution of a transformation), and data object. Applications access Chimeria functions via a Virtual Data Language (VDL), which supports both data definition statements (used for populating the database and for deleting and updating the virtual data definitions) and query statements (used to retrieve information from the database).

MCS. A design for a prototype Metadata Catalogue Service (MCS) for Data Grids is described in [161]. The MCS allows users to query based on attributes of data and provides management of logical collections of files and containers. The MCS prototype exclusively contains information that describes logical files, and assumes that physical file metadata is stored somewhere else. Examples of logical file metadata include information about how data files were created or modified, description of file contents, and file format information.

The design of MCS is based to a large extent on the Metadata Catalog (MCAT) of the Storage Resource Broker. Similar to MCAT, it allows specification of logical collection hierarchies, provides GSI authentication, and supports the notion of containers (a number of files managed together). Unlike MCAT, it holds metadata that describes only files and only for logical files.

The initial prototype is a simple, centralized metadata service. It uses the MySQL DBMS as the back-end, on top which a Web server with database connection is placed. The attributes in the current schema for the MCS can be divided into the following categories: Logical file metadata (logical file name, version number, container information, information about the creator and last modifier of the data), logical collection metadata (collection name, set of files that compose the collection, annotations on the collection, information about the creator and modifier(s) of the collection, collection hierarchy information), logical view metadata (view name, view attributes,
view creator and/or modifier, logical files, collections, sub-views within the view), authorization metadata (access privileges on logical files or collections), metadata that describes writers of metadata, audit metadata (actions performed via the metadata service), user-defined metadata (user defined attributes on logical files, logical collections and logical views), annotation metadata (descriptions of logical files, collections, views), transformation history metadata (information about how a logical file was created and what subsequent transformations were performed on the data), and finally external catalog access metadata (information needed to contact external metadata catalogs).

MDMS. The Metadata Management System (MDMS) [162] aims at providing a uniform interface to data intensive applications and Hierarchical Storage System (HSS). Applications and HSS can communicate with the MDMS to obtain high performance I/O from the underlying architecture. The environment leaves the task of choosing appropriate I/O techniques to the MDMS. The initial implementation targets only scientific codes that use large multi-dimensional arrays. The main functions of MDMS are the following: 1) Storing information about the abstract storage devices (ASDs) that can be accessed by applications; 2) Storing information about the storage patterns and access patterns of data sets; 3) Storing information about the pending access patterns; and 4) Keeping metadata for specifying access history and trail of navigation.

The components of the three tier architecture are the user application, the MDMS, and a hierarchical storage system (HSS). The MDMS is built using object-relational database technology. It keeps both system level and user level metadata about the data sets, data files, ASDs, physical storage devices (PSDs), access patterns, and users. MDMS communicates with user applications through directives. The main idea behind using directives is to help the system match the access pattern (i.e. how the data is accessed) and the storage pattern (i.e. how the data is stored), hence enable a specific I/O optimization in order to reduce the I/O bottleneck. Applications provide different types of information to MDMS using several different directives, including storage pattern directive, access pattern directive, abstract storage directive, I/O type directive, sequentiality directive, repetition directive, usage pattern directive, association (abstract data set space) directive, data set size directive, request size directive, and metadata query directive. The HSS has two main tasks in the environment: 1) Keeping the storage related metadata up-to-date in the MDMS; 2) Accepting the I/O optimization requests from the MDMS and I/O requests from the user application, and returning results to the application. The HSS employs a number of I/O optimization techniques, including among others, parallel I/O, collective I/O, sequential prefetching, data migration, and pre-staging. Hence, user applications give hints to the MDMS about the data access activities that they will perform using the directives, upon which the MDMS decides a suitable optimization technique and requests the HSS to perform that optimization technique when handling the user request.
Virtual Sky. Virtual Sky [32] is a portal to images of the night sky. Users can zoom out so that the entire sky is on the screen, or zoom in to a magnification of 2000. The portal also provides a gallery of interesting places, and a bulletin board where users can record comments.

The architecture is based on a hierarchy of pre-computed image tiles, where position in the sky is used as index. Multiple themes are possible, each one being a different representation of the night sky. The largest of the available themes is Digital Palomar Observatory Sky Survey (DPOSS) *. Images from all themes are resampled to the same standard projection, so that the same part of the sky can be seen in its different but aligned representations. The image storage and display is based on the Terraserver †.

Current work is on how to integrate the DPOSS and Sloan Digital Sky Survey ‡ data into a unified, distributed data set between two sites. Possible extensions include linkage of resampled, compressed images in Virtual Sky back to the raw data from the origin survey. The original images can be further processed using remote machines available through Grid resources to deliver custom products.

3.4.5 Other Systems

This subsection summarizes the systems that could not be classified in one of the previous groups; either because they offer very limited functionality to be included in a group, or because they offer a broad range of functionalities to be fit into one group. The only system classified here is the Enter The Grid portal. Although Enter The Grid offers a very limited set of functionality, it is included in the evaluation because it is a representative example for ‘single entry point’ type of portals.

Enter The Grid. Enter The Grid [33] is a portal providing a directory of resources on Grid computing. The directory includes company profiles (Grid technology providers, integrator providers, consultants, software and hardware providers), catalogs for Grid-enabled software and Grid services, and repositories for user groups, research projects and information sources.

3.5 Evaluation of the Studied Systems

This section presents the evaluation results. In total, 28 systems were studied and evaluated. The distribution of these systems among the classification groups (as described in Section 3.3) is as follows: 5 context-oriented systems, 10 computation-oriented systems, 6 instrumentation-oriented systems, 6 metadata-oriented systems, and 1 other more restricted system.

These systems were evaluated with respect to the evaluation criteria that were described in Section 3.2. The results of evaluation are presented in the form of tables.

*http://www.astro.caltech.edu/~george/dposs/
†http://terraserver.microsoft.com
‡http://www.sdss.org/
One table is created for each of the four evaluation criteria groups as described in Section 3.2. These tables are presented in Appendix C. Each system is evaluated with respect to each criterion. Evaluation of a system with respect to each criterion assumes one of the following values: ‘\(\checkmark\)’ (i.e. positive), ‘X’ (i.e. negative), ‘\(-\)’ (i.e. unknown). The following describes the evaluation of Zoo against three of the criteria:

1. Criterion: ‘Does the system provide experiment template management functionality?’
   Answer: ‘\(\checkmark\)’

2. Criterion: ‘Does the system identify/mention collaboration requirements?’
   Answer: ‘X’

3. Criterion: ‘Was a scripting language used for the development of the system?’
   Answer: ‘\(-\)’

Specifically, Tables C.2 and C.3 in Appendix C present the evaluation results with respect to the identified/mentioned requirements. Tables C.4 and C.5 present the results of evaluating the systems with respect to the functionalities that they provide. Results of the evaluation of the systems with respect to the applied enabler are presented in Table C.6. Finally, results of the evaluation with respect to the employed implementation technologies are given in Table C.7.

The remaining of this section provides an interpretation of these results and their analysis against the evaluation criteria defined in Section 3.2. The general conclusions, however, are provided in Section 3.7.

**Analysis of Results Against the Identified Requirements**

Against the infrastructure requirements, most systems identify/mention some requirements, while none of the systems identify/mention all requirements, and only Enter The Grid does not identify any of the infrastructure requirements. As can be expected, all context-oriented systems mention software environment requirements (because these systems target an integrated software environments that include a wide range of tools), all computation-oriented systems mention computing infrastructure requirements, all instrumentation-oriented systems mention instrumentation requirements, and all metadata-oriented systems mention storage infrastructure requirements. Software environment requirements are mentioned also by all computation-oriented systems, since these systems aim to provide tools for the management of resources; in addition, some systems in this category specifically develop (analysis) tools for a given science domain (e.g. problem solving environments including Cactus and Ecce).

With respect to the interface requirements, requirements related to user interfaces are mentioned by almost all systems, except Cactus, Chimera, MDMS and Virtual Sky. First three systems offer their functionality either through command line interfaces or by being integrated in other systems (e.g. Cactus is used by ASC).
Virtual Sky provides an archive of night images of sky. Thus, user interfaces must be a topic of interest for this system, however, it is not explicitly mentioned in the literature about this system. There seems no rule in identifying/mentioning *programming interfaces requirements*, systems randomly mention these requirements.

Regarding the **identified functionality requirements**, all context-oriented and computation-oriented systems mention *experiment management functionality requirements*. Metadata-oriented systems generally do not mention these requirements, since they only deal with the metadata (and in some cases with the data). Unexpectedly, instrumentation-oriented systems do not mention *experiment management functionality requirements* either. *Requirements for information management functionality* are mentioned by all context-oriented and metadata-oriented systems; while *resource management functionality requirements* are mentioned by all computation-oriented systems (because these systems aim to utilize and manage available resources for high-performance computing) except FAST, which is mainly addressing visualization and collaboration issues of computing. Systems mention the remaining requirements randomly.

Similarly, **implementation requirements** are randomly mentioned by the evaluated systems.

**Information management requirements** are mainly mentioned by context-oriented systems and metadata-oriented systems. Ecce and CLRC Data Portal identify and mention all information management requirements. Computation-oriented and instrumentation-oriented systems do not mention information management requirements, with the following exceptions: UNICORE which models and manages jobs, BioPSE which defines new data structures for dynamic optimization and integration of code, ASC which supports (some kind of) virtual organizations and access rights, Ecce which was originally designed at the object integration level (hence everything is represented as an object in the database), and Virtual NMR Facility which supports storage of metadata (conclusions) into electronic notebooks. Among those systems that identify information management requirements, many mention mainly the *modelling, storage and manipulation requirements*.

**Analysis of Results Against the Provided Functionalities**

Among the studied systems, only Sieve/Symphony provide all *experiment management functionality*, while Zoo and PELLPACK support all except one; it is unknown from the literature whether Zoo provides session management or PELLPACK provides metadata management. As can be expected, context-oriented systems provide most of the *experiment management functionalities*, computation-oriented systems focus on the *design and execution of experiments* (i.e. jobs), instrumentation-oriented systems provide *experiment execution functionality* (i.e. running an instrument), and metadata-oriented systems provide *metadata management functionality*. Only Zoo, PELLPACK, Sieve/Symphony, OPM, and Ecce provide *experiment templates* to their users; while Virtual NMR Facility, Virtual Geotechnical Lab, VCLab, and all context-oriented systems provide *assistance* to scientists during (at least one) stage of experimentation. Two of the reasons for these systems to provide assistance
is either because they are used for educational purposes (Virtual Geotechnical Lab, VCLab) or because they offer training to scientists (SoftLab, Virtual NMR Facility).

**Data/information management functionalities** are ‘partially’ provided by the evaluated systems. Context-oriented systems commonly provide *data/information management functionality* for experiment results (as part of the experiment) and *query functionality*, while metadata-oriented systems commonly support querying (of the metadata catalogues). Remaining systems, computation-oriented and instrumentation-oriented, do not provide any ‘visible’ functionality for *data/information management*. Exceptions to this are UNICORE (which provides data file management functionality as input and output for the experiments), Cactus and ASC (which provide partially Grid-based I/O libraries), Ecce (which support management of data sets and metadata in an experiment and querying of this metadata), Virtual NMR Facility (which allows scientists to save their conclusions and some files in electronic notebooks), and Virtual Geotechnical Lab (which guides users during the interpretation of results). Access to data/information from heterogeneous sources (supported by Zoo, OPM, CLRC Data Portal, SRB and Virtual Sky), data/information integration (supported only by OPM), version control (supported only by CLRC Data Portal) and replication (supported by none of the systems) constitute the functionalities that are provided by only a small section of the evaluated systems. On the other hand, replication is a popular topic among the Data Grid related projects (see Subsection 3.1.3).

Different **resource management functionalities** are provided by the evaluated systems. Among context-oriented systems, only Zoo (*monitoring*) and PELLPACK (*discovery, allocation, scheduling and monitoring*) provide resource management functionalities. Both systems consider software tools as resource; Zoo additionally refers to instruments as resource. For computation-oriented systems, resources are mainly hardware (computing) resources. Resource management in these systems is supported by the Grid. ASC is the only system that provides all resource management functionality to its users. For instrumentation-oriented systems, resources refer to instruments and monitoring (of the instrument operation) is the commonly provided functionality. For metadata management systems, resources are data sets, for which *discovery mechanisms* are provided through querying of the *descriptive information* about the data sets.

There is little information in the literature about the **user management functionality** provided by the evaluated systems. *User accounts* are created and managed by the computation-oriented systems that are based on Grid, by some of the instrumentation-oriented systems that require users to possess certain privileges or acquire time slots for instrument operation (Virtual NMR Facility, Virtual Lab, Virtual Geotechnical Lab), and by metadata-oriented systems that keep track of data provenance and history of activities in the metadata for data sets. Similarly, *role management functionality* is provided only by a few systems (ASC, SRB, MCS).

Among **collaboration-related functionalities**, resource sharing is the functionality that is widely provided. Context-oriented systems allow scientists to share experiment designs and sometimes results, computation-oriented systems allow users to share computing resources (and analysis software shared by those systems where
experiments are ‘composed’ from executables), instrumentation-oriented systems allow the sharing of instruments, and metadata-oriented systems allow the sharing of data sets. The most comprehensive set of collaboration functionality is provided by ASC, which allows its users to share (computational and software) resources, to collaboratively develop software programs, and to control the collaboration through virtual organizations support. Virtual NMR Facility allows multiple scientists to share a session for operating an instrument, view the same data sets, and discuss the results (e.g. using whiteboards).

For many systems, there was no mention of security in the literature. Grid-based computation-oriented systems provide authentication/authorization and single sign-on functionalities, while metadata-oriented systems provide all security related functionality.

**Analysis of Results Against the Applied Enabling Technologies, Standards and Paradigms**

As can be seen from Table C.6, the most common “enabler” used by the evaluated systems is the Grid technology. Many systems already make use of Grid facilities, and still many of the remaining are planning to port their systems to the Grid environment (e.g. Ecce and Virtual Sky).

The remaining enablers, however, are not as widely applied by the evaluated systems. Among all systems, only Sieve/Symphony, Ecce, CLRC Data Portal and SRB have addressed modelling issues and applied a standard information model, only CLRC Data Portal, SRB and Virtual Sky have applied distributed information management, and only ASC has applied the virtual organizations paradigm.

None of the evaluated systems use workflow management systems. However, PELLPACK, Sieve/Symphony, Gateway, SCIRun, BioPSE, and Uintah allow their users to compose their experiments by attaching a number of components to each other and creating a data flow between the connected components. PELLPACK and Sieve/Symphony additionally create a control flow between the components, and perform type checking for the components connected to each other.

**Analysis of Results Against the Employed Implementation Technologies**

Unfortunately, implementation issues were not discussed in details in the surveyed literature.

Almost all systems support Unix environment. Only Cactus and ASC are available for a number of other platforms, including Linux and Windows.

Similarly, C/C++ are still the preferred languages for implementation, although Java is gaining popularity. Java is being used especially for the implementation of client software (user interfaces), as well as for the implementation of server side components (specifically JavaBeans, RMI, Servlet technologies are used). FORTRAN is used to support especially existing numerical analysis software. Some systems also use scripting languages for the implementation of user interfaces or for the automation of
regularly performed tasks, such as substituting parameterized values in a configuration file with actual values.

An internal DBMS is employed by almost all systems. The majority of the used DBMSs are relational and file-based. CLRC Data Portal uses Native XML DBMS while MDMS uses an object-relational DBMS.

Although older systems provided only command line interfaces to users, Web-based graphical interfaces are becoming available for many systems, resulting in a wider acceptance of Web technologies such as Web servers and Servlet or similar technologies. In many systems, authentication is performed by modules plugged-in to Web servers (e.g. UNICORE). XML is used by Gateway for representing resource descriptors, by CLRC Data Portal as the main data representation and exchange format, and by Ecce for data exchange.

3.6 Evaluation of the Proposed VLAM-G/VIMCO Approach

The general evaluation framework introduced and described in this chapter and the evaluation criteria described in Section 3.2 can be used for studying any given support environment and its information management platform in details. Therefore, the framework can also be applied to study and evaluate the proposed VLAM-G/VIMCO approach.

VLAM-G and VIMCO were introduced earlier in Subsection 1.5.1. In order to compare the proposed scientific information management platform with the information management approaches of the related work, VIMCO is evaluated using the same framework. Similar to the other evaluated systems, the entire VLAM-G is used for the evaluation rather than only its information management platform (i.e. VIMCO).

Evaluation and comparison results of VLAM-G/VIMCO can be found in the tables presented in Appendix C, together with the evaluation results of the other studied related systems. However, detailed evaluation and comparison of the VLAM-G/VIMCO is provided in conclusions in Chapter 7.

3.7 Conclusions

Scientific experiments are complex, both in terms of their designs and data/information that they generate. Several systems exist and are emerging with the purpose of supporting scientists during their experimentations. This chapter first presented the technologies, paradigms and standards that enable the development of such systems. Secondly, the most representative examples of the related work were classified into four categories based on which aspect of experimentation that they focus on, where the aspects are defined as context, computation, instrumentation, and metadata. List of evaluation criteria were defined for the evaluation and comparison of the systems in the related work. For each system, a brief summary was presented. Finally, evaluation results were presented and interpreted. The proposed VLAM-G/VIMCO
approach was also evaluated with respect to the same criteria (the evaluation and comparison results for VLAM-G/VIMCO are presented in Chapter 7).

The following constitute the global conclusions that can be drawn from the study and evaluation of the related work presented in this chapter:

- The requirements identified in Chapter 2 represent the wide spectrum of functionality needed for a proper support of scientific experiments and applications. However, each of the evaluated systems only provides a part of the required functionality. As their classification implies, context-oriented systems provide extensive information about the contexts of experiments, by focusing on experiment designs and history of activities; computation-oriented systems provide the necessary environment for defining computations and the necessary facilities for their executions; instrumentation-oriented systems provide the functionality for controlling/operating a physical or simulated instrument; and metadata-oriented systems provide catalogues for keeping information about data sets. A comprehensive system that provides support for all these functionalities and presents them to scientists in an integrated environment is missing. The key point here is the integrated support environments that enable one-stop experimentation.

- The enablers presented in this chapter constitute a strong basis for the development of support environments. However, except the Grid technology, the enablers are not yet fully applied by the existing systems. Nevertheless, even the systems that are built on the Grid technology do not provide adequate functionality, mainly because the Grid is currently targeting compute-intensive applications. Once mature enough, facilities that will be provided by the Data Grid must be incorporated by the systems. Combination of the powerful process control mechanisms of workflow systems, distributed information management and integration mechanisms of federated database systems, controlled and organized collaboration mechanisms provided by virtual organizations, and the distributed resource management facilities of Grid and Data Grid will constitute the strong steps towards the realization of integrated support environments.

- Currently the main focus in the area of support environments is on the utilization of available resources using the Grid facilities, approaching from the technological point of view. Conceptual aspects of experiment management, such as modelling experiments, defining experiment templates, providing assistance to users, modelling and management of experimental information did not receive much attention. Those facilities provided by the experiment management systems are the only examples, which need to be further extended and improved to fully support the identified requirements.