Chapter 4

Information Management Framework for CEE

As described in Chapter 1, e-science applications have many challenging characteristics, which make it difficult for scientists to perform their experiments. Several different approaches for environments supporting the scientific experimentation were described in Chapter 1, among which the virtual laboratory (VL) was proposed as a solution.

The VL solution was further characterized in Section 2.2, and different types of VL users and the activities that they can perform within the VL were identified and described in Section 2.3. As explained, each of these users has different needs and expectations from the VL environment. These user requirements impose a number of requirements on the base ICT infrastructure that underlies the VL environment. During the requirements analysis presented in Chapter 2, particular attention was given to information management requirements (as the aim of this thesis) in addition to general VL requirements.

As these requirements pointed out, there are several different aspects to be addressed and focused on the management of experiment-related information, which include among others modelling the scientific experiments as well as the data and functionality modelling for information about the experiments. Further, the analysis of requirements showed that many of the requirements are inter-related with each other, thus the VL solution must be approached as an integrative solution. For example, while focusing on the management of information in a VL, other aspects of the solution must also be considered, such as the interfaces, resource management, user management, collaboration, etc.

In Chapter 3, the results of an extensive study and evaluation of several representative examples and state-of-the-art in different supporting environments for scientific experimentation were presented. The evaluation of related systems showed that these systems address and propose solutions for only certain aspects of scientific experimentation and do not provide a comprehensive solution system. For instance, UNI-
CORE approaches experiments as jobs that need to be executed on high-performance computing systems, while OPM provides data structures for representing scientific experiments but it does not address experiment execution.

Hence, as can be concluded from this study and evaluation, there is a need for

integrated solutions and support environments that address every aspect of experimentations and that support scientists during the entire life cycle of experiments

which are referred to as Collaborative Experimentation Environments (CEEs) in this dissertation. This definition of a CEE is in accordance with the definition given in Subsection 1.2.2, where a CEE was defined as a virtual laboratory in its broadest sense (please refer to Subsection 1.2.2 and Figure 1.1 in that subsection).

Consequently, although the focus of this chapter is on defining an information management framework for CEEs, the chapter also addresses several other aspects of a CEE, mainly the experiment model and the user environment. Structure and classification of the framework for management of information in CEE is outlined in Figure 4.1. This chapter first describes an experiment model for CEE that is proposed as a base for uniformly representing heterogeneous scientific experiments. Then a framework for the user environment of a CEE is introduced, which describes how data handled in the CEE and functionality provided by the CEE can be presented and provided to users. This chapter then focuses on the information management framework for CEEs, by providing data models for the representation of experiment-related

Information Management Framework for CEE

- Modelling Scientific Experiments
  - Approach used for modelling scientific experiments
  - The proposed experiment model

- User Environment of CEE
  - Facilities provided by the user environment
  - Graphical representations of experiments in the user environment

- Data Modelling for Information Related to Scientific Experiments
  - Approach used for modelling experiment-related information
  - Data modelling for information about scientific experiments
  - Data modelling for information about users
  - Data modelling for CEE-related information

- Functionality Modelling for Managing Information Related to Scientific Experiments
  - Functionality for managing information about scientific experiments
  - Policies for managing experiment contexts
  - Functionality for managing scientific data/information

- Functionality Modelling for Collaboration, Administration, Security
  - Collaboration functionality
  - Administration functionality
  - Security functionality

- Methodology for Integrating New Domains and Applications in CEE
  - Integration of a new domain in CEE
  - Integration of a new application in CEE

Figure 4.1: Structure and classification of the framework for management of information in CEE
information and *functionality models* for the management of this information. Please notice that a wide variety of information related to scientific experiments needs to be modelled in CEE, including the information about the experiments (e.g. descriptions of experimental procedures and computational processes), information about the users (e.g. user information and access rights definitions), and CEE-related information (e.g. information on sessions). Finally, a *methodology* is defined and described for integrating new domains and new applications in a CEE, showing the genericness and reusability of the results presented in this chapter.

### 4.1 Modelling Scientific Experiments

In an experimentation environment, it is natural that all activities are centered around *experiments*. User requirements presented in Chapter 2 emphasize the need for mechanisms to uniformly manage various types of experiments. In specific, identified requirements included availability of mechanisms and facilities for designing experiments that clearly and sufficiently represent the experiment objective, for executing experiments, and for managing information generated by experiments. As also mentioned as a base infrastructure requirement in Chapter 2, a clear definition of a scientific experiment must be provided in order to properly address the user requirements.

This section focuses on the modelling of scientific experiments and proposes an experiment model. The experiment model underlying a CEE outlines the overall approach that the CEE follows for supporting scientific experiments. The experiment model has a major influence on how the functionality and facilities of the CEE are presented to users, how users interact with the CEE, and how experiment descriptions as well as data/information are stored and managed. Furthermore, the experiment model allows a methodological definition of complex experiments in a problem domain. Thus, the experiment model lies in the core of a CEE.

In the remaining of this section, first the approach used for scientific experiments is outlined, then the experiment model proposed for CEEs is described.

#### 4.1.1 Approach Used for Modelling Scientific Experiments

In this subsection, the approach followed for systematic modelling of scientific experiments is described. A summary of the approach is given in Figure 4.2. The steps involved in the approach include: (1) performing a detailed analysis of different types of scientific experiments in different domains and identifying the process and data flow in each experiment type, (2) cross-studying the process and data flows, identifying the different components of experiments, and generalizing to achieve a high-level general structure of scientific experiments, (3) further studying the experiments and generalizing to define an experiment model. Please notice that Figure 4.2 provides only a glance over the approach. In the following subsections, this approach and results of each step are described. Furthermore, all the elements in the figure are described and depicted in larger figures. However, first an illustrative example experiment is presented, which will be used to exemplify the activities when describing the steps.
Step 1

**Study different types of experiments**
Identify the process and data flow in each experiment type

**Process Data Flow (PDF)**
for each experiment type

Step 2

**Cross-study the PDF for each experiment**
Identify different types of components involved in the experiments
Generalize and define a high-level, general structure for scientific experiments

Step 3

**Study/identify key aspects of scientific experiments (e.g. definition, description, execution)**
Generalize and define an experiment model by mapping these aspects to the components of the experiment model

**Experimen t model**

Figure 4.2: Overview of the approach used for modelling scientific experiments

**An Illustrative Experiment Example**

The example experiment is based on a simplified version of microarray experiments. In this example experiment, a microarray containing a number of clones is hybridized with mRNA probes that are extracted from a sample. Every clone that is spotted on the microarray represents a gene for an organism. The result of the hybridization is the hybridized array. The hybridized array is scanned by a laser scanner to obtain an image of the hybridized array. This image consists of spots with different color intensities, where a spot corresponds to a clone on the microarray. The image is analyzed using an image analysis program to quantify the intensities of the spots. The result of the experiment is this collection of intensity values for each spot.
Process Data Flow as a Tool for Modelling Experiments

The first step in the approach was to perform a detailed study of different types of scientific experiments in different domains (which include among others microarray experiments, material analysis experiments, traffic simulation studies, and confocal microscopy experiments). The main results of this study step were comprehensive step-by-step definitions of experiments, each represented as a conceptual model. In this conceptual model, every component involved in an experiment (e.g. activity, data, physical entity), its properties, and relationships to other components were defined. Such a detailed and comprehensive definition of an experiment is referred to as the Process Data Flow (PDF).

General Structure of Scientific Experiments

Having the PDFs for different types of experiments, the second step in the approach was to cross-study the PDFs and identify the different types of components involved in the experiments.

Experiment Components. As the cross-study of PDFs showed, during a typical scientific experiment:

- **input** is taken, where the input can be a physical entity (e.g. a microarray) or data (e.g. image of the hybridized array);
- an **activity** is performed on the input, where the activity can be a laboratory activity (e.g. hybridization), an instrumentation (e.g. scanning the hybridized array), or a computational process (e.g. analyzing the image of the hybridized array);
- **output** is generated, where the output can be a physical entity (e.g. hybridized array) or data (e.g. intensity measurements for spots on the microarray).

Therefore, experiments are composed of three kinds of experiment components, corresponding to the different types of input, activities, and output (see Figure 4.3):

1. **physical entity** (typically a part of the input to an activity in an experiment as well as a part of the outcome of an activity in an experiment),
2. **activity** (typically laboratory activities, instrumentations, and computational processes involved in an experiment), and
3. **data element** (typically either used as a part of the input to an activity in an experiment, or generated as a part of the output of an activity in an experiment).

**Physical entities** are used during laboratory activities or during instrumentation. Figure 4.4 depicts the experiment components involved in the example microarray experiment. In this example, microarray, hybridized array, sample and mRNA probe constitute the physical entities.
**Activities** may represent laboratory activities, instrumentations, or computational processes. Examples of activities in Figure 4.4 are, respectively, hybridizing a microarray with mRNA probes from the treated sample (i.e. hybridization), scanning the microarray with a laser scanner device (i.e. array scanning), and analyzing the array image using special analysis software (i.e. array image analysis).

**Data elements** in turn correspond to both raw data generated by instruments and used as input to computational processes (e.g. array image in Figure 4.4), as well as processed data and/or information generated by computational processes (e.g. array measurement in Figure 4.4). Generic descriptive elements constitute a special type of data elements. The generic descriptive elements represent components that are common to all experiments and provide descriptive information about them. Such descriptive information does not change from one experiment to another. In Figure 4.4, clone, gene, organism, scanner and image analysis program are generic descriptive elements. For example, scanner represents the device used to scan microarrays and produce images. This generic descriptive element provides descriptive information about the device, such as its vendor, model, parameters, etc. Usually this information is provided once (e.g. by a domain expert or administrator), and used by scientists in their microarray experiments.

Once the different types of experiment components were identified, the PDFs defined during the first step were represented using these experiment components. Figure 4.5 presents the microarray PDF as an example using the same notation as Figure 4.4 for experiment components. This example microarray PDF depicts the order of components in the experiment (i.e. the experiment flow) and also the relationships between components and generic descriptive elements in microarray experiments.

**High-Level General Structure of Scientific Experiments.** Further studying and generalizing the relationships between the components in an experiment as well as the relationships between experiments, a high-level, general structure for scientific experiments was achieved.

As illustrated in Figure 4.6, a scientific experiment consists of a number of experiment components. As described earlier, components of an experiment correspond to

![Experiment Components](image)

Figure 4.3: Different types of experiment components
activities, physical entities, or data elements involved in that experiment.

Every experiment is part of a project. In a project, there may be one or more experiments that are inter-related. Experiments in a project may follow an order (e.g. time course experiments), or may be unordered (e.g. comparison experiments). A general description is associated to each project, providing overall information about the project such as the (common) objectives of the experiments involved in the project.

The detailed studies (as described above) were complemented with the experiment characterization and requirements analysis (as presented in Chapter 2) to identify the key aspects of scientific experiments (e.g. definition, description, execution). These form the building blocks of the experiment model, which is described next.

### 4.1.2 The Proposed Experiment Model

The proposed experiment model consists of three main components [163], namely experiment procedure, experiment context and experiment's computational processing (see Figure 4.7).

An experiment is defined by its procedure and described by its context. An experiment procedure defines the approach taken to solve a particular scientific problem, by defining the experiment components that are typically involved in the experiments of the same type (i.e. experiments that address the same scientific problem). Thus, an experiment procedure standardizes the experimental approach for experiments of the same type. On the other hand, an experiment context describes the solution. An experiment context is an instantiation of an experiment procedure. It describes the accomplishment of a particular experiment, by providing descriptions of each experiment component involved in the experiment, in other words, by providing the context

---

![Figure 4.4: Experiment components for the example microarray experiment](image)
for each experiment component. Thereby, it provides the context for that particular experiment. Due to the need for processing of large data sets, computational processes constitute an important part of scientific experiments. The computational processes involved in a scientific experiment are collectively called as the experiment's computational processing.

Components of the proposed experiment model are described in details below.

**Experiment Procedure**

An experiment procedure defines a particular way of accomplishing an experiment of a certain type. It represents the (steps of the) experimental approach taken by a scientist to solve a specific scientific problem. In other words, an experiment procedure step-by-step defines how to make a certain type of experiment. The complete definition of an experiment includes, among others, the components involved in the experiment, their order, and the control variables and parameters as well as the protocols to be applied during the experiment.

The same type of experiments typically involve the same components (i.e. activities, physical entities, and data elements). Furthermore, these components appear

![Figure 4.5: Process Data Flow for microarray experiments](image-url)
in the same order in successive experiments. An experiment procedure is composed of elements that correspond to such experiment components. A procedure element is not an actual experiment component, rather a placeholder for the component. There is one procedure element for each experiment component. Similar to representing experiment components by procedure elements, an experiment procedure also represents how the components in an experiment are related to each other through corresponding relationships among procedure elements (e.g. the order among the experiment components). Therefore, by defining the components that are typically involved in the same type of experiments, an experiment procedure standardizes the experimental approach for that type of experiments.

Figure 4.8 illustrates a procedure for the example experiment. As mentioned above, the elements of this procedure correspond to the components involved in the example experiment. For instance, microarray procedure element corresponds to the microarray physical entity. Note that the generic descriptive elements (e.g. scanner) are similarly represented by generic descriptive procedure elements in this example procedure. As also mentioned above, the procedure includes the relationships between the procedure elements, such as the order of the procedure elements (i.e. the experiment flow), and the relationships between the procedure elements and the generic descriptive elements.

From one point of view, an experiment procedure can be seen as a template for a certain type of experiment, since it represents both the experiment components that are common to successive experiments and the relationships among them. However, in the proposed experiment model, a procedure means more than a template. Experiment procedures are defined by scientists that have extensive knowledge and experience in the scientific domain of the experiments (i.e. by domain experts). Therefore, a procedure captures the expertise and knowledge of the expert in that domain, and serves as a facilitator for preventing the loss of expertise and knowledge by transferring them to the users of the procedure. This way, experiment procedures can be used to provide assistance to novice users for complex experiments, helping them avoid making mistakes and increase the efficiency and accuracy of their experiments.

**Experiment Context**

An experiment context is an instantiation of an experiment procedure. While an experiment procedure defines the experimental approach taken to solve a scientific problem, an experiment context represents the solution. It describes the accomplishment of a particular experiment by describing each of the components involved in that experiment. Descriptions of the experiment components include the actual values that
are used for the variables and parameters, and any other information that is necessary to describe the activities performed, physical entities that are treated/handled, or data elements that are used or generated during the experiment. In other words, an experiment context provides the context for each experiment component, and hence the context for the entire experiment.

The example experiment context depicted in Figure 4.9 describes a particular microarray experiment to study the expressions of genes in a mouse tissue. It is an instantiation of the example experiment procedure given in Figure 4.8. In this example context, the procedure elements are instantiated, i.e. descriptions of the experiment components involved in this particular microarray experiment are provided. Due to its large size, details of some context elements (namely hybridization, hybridized array, array scanning and hardware) are omitted from the figure. Similar to the example procedure, this example context also includes elements corresponding to the generic descriptive elements (e.g. clone, gene, organism). Since a microarray contains thousands of clones and there is one gene for each clone, the context elements for clones and genes are depicted in multiple copies. Also similar to the example procedure, this experiment context provides the relationships between context elements (i.e. the experiment flow), and between context elements and generic descriptive context elements.

**Experiment’s Computational Processing**

Large size of generated data is one of the main characteristics of experiments and applications in e-science domains (see Section 1.1). Availability of a high-performance hardware and software infrastructure for the processing and analysis of these large data sets is among the major user requirements (see Section 2.5). Therefore, computational processes constitute an important part of scientific experiments, and hence require special attention. This subsection focuses on the computational processes and addresses their representation as part of the proposed experiment model. For simplicity, ‘computational processes’ will be referred to as ‘processes’ in the remaining of this subsection. Furthermore, existence of a software tool is assumed for each computational process (e.g. ScanAlyze for array image analysis).

In an experiment, data flows from one process to another during its processing.

![Experiment Diagram](image-url)
and analysis (e.g. from a clustering process to a visualization process). This data flow can be represented by a directed graph. Nodes in the graph are processes, while the connecting arcs represent the data flow through the processes. This data flow graph represents the experiment's computational processing. An experiment's computational processing consists of a number of processes connected to each other. A process may be performing an experiment specific task or a generic task. An example of experiment specific tasks is the control of a laboratory instrument, while file manipulation is an example of generic tasks. When defining an experiment’s computational processing, a user specifies the values for the required variables and/or parameters for each process. Since there is a software tool corresponding to each process, the variables and parameters are actually the variables and parameters required by the software tool. When defined, processes in the experiment’s computational processing

![Diagram of experiment procedure](image)

**Notation:**
- **NAME** Procedure Element
- **NAME** Generic Descriptive Procedure Element
- Experiment flow
- Relationship between procedure elements
- Relationship between two generic descriptive procedure elements

**Figure 4.8:** Example experiment procedure for the example microarray experiment
Figure 4.9: Example experiment context for a particular accomplishment of the example microarray experiment
are executed on one or more computers.

Figure 4.10 depicts a computational processing for the example microarray experiment, which consists of three processes, each with a corresponding software tool. This example processing aims to analyze the image of the hybridized microarray and quantify the intensity values of the spots on the microarray. For this purpose, the first process reads the image file from its location and transfers the image to the analyzer process, which quantifies the spot intensities and passes the intensity values to the last process. The last process writes the analysis results into a new file. File reader and file writer processes are examples of generic processes, while image analyzer is an experiment specific process.

As mentioned earlier, a (computational) process is a specific kind of activity, hence, it is an experiment component (see Figure 4.3). Therefore, an experiment procedure may contain elements that correspond to processes. Consequently, particular experiments that are instantiating this procedure contain descriptions of these processes in their experiment contexts. This is illustrated in Figure 4.11, where the procedure, the context and the computational processing for the example microarray are provided. Note that the figure contains the same procedure, context and computational processing for the example experiment that were given in Figures 4.8, 4.9 and 4.10, respectively. As can be seen in this figure, both the experiment procedure and the experiment context include elements that correspond to computational processes, which are encircled (i.e. array image analysis process and its software tool). Furthermore, the experiment procedure and the experiment context also contain elements that correspond to the input/output for the computational process, which are printed in bigger fonts (i.e. array image as input and array measurement as output). This experiment’s computational processing consists of three processes, corresponding to input generator (i.e. file reader reading the array image), process (i.e. image analyzer processing the array image), and output consumer (i.e. file writer storing the

Figure 4.10: Example computational processing for the example microarray experiment
array measurement raw data). Also note here that the context and computational processing of an experiment complement each other to provide information about that particular experiment. For instance, location of the array image that is required by the file reader is contained in the context (in specific, in the context element for the array image), while the experiment’s computational processing provides run-time information (e.g. the needed environment variables and their values).

4.2 User Environment of CEE

User environment is the users’ entry point to a CEE. Users interact with the CEE and perform their activities in the CEE through the user environment. Facilities and functionality provided by the CEE and its information management platform are also made available to users through the user environment. Therefore, there is a direct relationship between the facilities provided by the user environment and the facilities provided by the CEE and its components.

Figure 4.11: Procedure, context and computational processing for the example microarray experiment. Details can be found in Figures 4.8, 4.9 and 4.10
As pointed out by the requirements in Sections 2.5 and 2.6, user environment of a CEE must include graphical, easy and convenient to use, and flexible user interfaces. Furthermore, it must provide the necessary facilities to support users’ diverse activities during all stages of an experiment. In specific, the user environment must adequately support the capturing and presentation of different types of data/information managed by the information management platform of the CEE.

Considering the diverse activities that scientists perform within a CEE as well as different types of data/information handled in a CEE, reusability becomes one of the main goals of the user environment. Note that this reusability is not limited to software reusability. It also includes reusability of experience, information and knowledge, techniques, and available hardware and/or software resources. The experiment model described in Section 4.1 forms a strong base to achieve such reusability.

In the following sections, CEE user environment is described, focusing on its facilities and the presentation of experiments to users.

### 4.2.1 Facilities Provided by the User Environment

The experiment model presented in Section 4.1 directly influences the facilities provided by the user environment of a CEE. The component-oriented nature of the experiment model allows graphical representation of its components. The component-oriented nature and the graphical representability of the components of the experiment model are used as the base for the design of the facilities provided by the user environment. Facilities provided by the user environment of the CEE consist of three graphical editors, namely Procedure Editor, Context Editor, and Processing Editor. These editors correspond to the components of the experiment model and provide the base functionality needed to support users.

The **Procedure Editor** supports the base functionality for experiment procedure management, such as designing new experiment procedures and viewing and/or modifying existing experiment procedures. An experiment procedure defines an experiment from a specific problem domain, thus the Procedure Editor is available only to domain experts who have extensive knowledge and expertise in that problem domain. The Procedure Editor can be used to provide further help to domain experts. For instance, when designing a new experiment procedure for a domain, a domain expert can make use of the schema elements of the application database for that domain to represent the experiment components in the procedure. Furthermore, the Procedure Editor may provide utilities to ease the procedure management, such as automatically capturing and storing the graphical properties of the procedure being defined by the domain expert. A domain expert can modify an existing experiment procedure by selecting the procedure from a list of available procedures.

The **Context Editor** is an editor for instantiating procedures (i.e. for generating the experiment context) and for viewing and/or modifying existing experiment contexts. The Context Editor is mainly used by scientists. If a scientist wishes to make a new experiment, the Context Editor presents the list of available experiment procedures that s/he can use. The experiment procedure selected by the scientist is displayed in the editor window, based on the graphical properties defined and saved
by the domain expert along with the procedure. The scientist then creates a new context for the experiment by providing a description for each element in the procedure. The resulting experiment context contains descriptive information about the experiment being performed. Similar to the list of available experiment procedures, a list of existing experiment contexts is presented to the scientist if s/he wants to view and/or modify an existing experiment context. In this case, all information associated with the experiment context selected by the scientist is loaded from the database and presented to the scientist in the Context Editor. Necessary information for the proper display of the context is obtained from the corresponding experiment procedure that is used as template when creating this context. Similar to the Procedure Editor, the Context Editor can be used to further support scientists by providing more functionality, for instance for querying the database to retrieve the context of an existing experiment component, or for guiding and/or assisting scientists during the instantiation of the experiment procedure.

The Processing Editor allows a scientist to define a data flow for the required computational processing in her/his experiment. A processing is defined by attaching a number of software tools to each other, each of which performs an activity on the input data and generates output data. A computational processing can be saved for future references. The Processing Editor is initiated automatically when the scientist instantiates an element in the experiment procedure that corresponds to a computational process and marked as a ‘processing element’. Besides the base processing management functionality, the Processing Editor can be used to further support users, for instance, during the design of a new computational processing by displaying the descriptions of available software entities to users. The description provides, among others, information about the run-time requirements of a software entity (e.g. CPU and memory requirements).

4.2.2 Graphical Presentation of Experiments in the User Environment

Uniformity is the key for presenting experiments in the user environment of a CEE. All experiments and experiment management functionality supported by the CEE user environment must be presented to users in a uniform manner, despite the diversity of possible scientific experimentations, heterogeneity of the data/information handled by these experiments, and complexity of experiments.

Uniformity is mainly achieved by the proposed experiment model and the data models for the components of the experiment model. The data models will be described in Section 4.3. Another aspect supporting uniformity is the component-oriented graphical representations of procedures, contexts, and processing of experiments. The component-oriented graphical representations use workflow-like structures with nodes and arcs among nodes, where nodes represent experiment components and arcs represent data and/or control flowing between components.

Uniform graphical presentation of experiments in turn allows uniform presentation of experiment management functionality to users. Uniformity in functionality presentation is achieved through the three editors, that were described in the pre-
vious subsection. All editors provide the base functionality of writing, reading, and displaying of information uniformly, mostly through operations on graphical elements such as selectable lists, menu items, and drag-and-drop operations.

Another important aspect of the user environment is to guide and assist users during experimentation. Guidance can be provided to users through descriptions of experiment components represented as experiment procedures, and assistance can be provided through context-sensitive information saved along with an experiment procedure and used when instantiating the procedure. For instance, the Context Editor can provide assistance to users in the form of context-sensitive menus displayed by mouse-clicks on procedure elements.

4.3 Data Modelling for Information Related to Scientific Experiments

Modelling experiment-related information handled in a CEE is important, since the employed data models have a direct influence on the CEE and its functionality.

As mentioned in Sections 2.5 and 2.6, modelling wide variety of information related to scientific experiments is among the major requirements. Information related to scientific experiments handled in a CEE includes information about experiments themselves (e.g. experiment procedures, experiment contexts, computational processing), information about users (e.g. information about users and user roles, access rights definitions), and other CEE-related information (e.g. session information, information about the services provided by the CEE). Another main requirement for information management is the availability of a methodology for modelling and management of experiment-related information to prevent the duplication of work.

This section builds on the experiment model and the user environment as presented earlier in this chapter, and describes the modelling of information related to scientific experiments in a CEE. It first describes the approach taken for modelling information related to scientific experiments. Then the specific data models are described, including data models for information about experiments, users, and the CEE itself.

Note here that all data models described in this section are object-oriented, and they are illustrated using UML diagrams.

4.3.1 Approach Used for Modelling Experiment-Related Information

In this subsection, the approach followed for systematic modelling of experiment-related information handled in a CEE is described. A summary of the approach is given in Figure 4.12. The steps involved in the approach includes: (1) performing a detailed analysis of the PDFs for different types of studied experiments, identifying the most common aspects among them, (2) developing a data model with generic constructs to represent these common aspects, and (3) reusing the PDFs to extend the generic model for developing application specific data models. Please notice that
Chapter 4. Information Management Framework for CEE

Figure 4.12: Overview of the approach for modelling experiment-related information

Analyzing Process Data Flows

Process Data Flow (PDF) was introduced in Section 4.1 as a tool for modelling scientific experiments. Recall that the approach used for modelling experiments described in that section included a step for studying different types of experiments which gen-
erated a PDF for each type of experiment. PDF is also used for modelling experiment-related information. The first step in the approach used for modelling experiment-related information was to perform a detailed analysis of the PDF for each type of studied experiments. The focus of the analysis was to identify the commonalities among the different types of experiments.

**Common Aspects of Scientific Experiments**

In order to provide *generic* solutions to support experiments from different e-science domains, common aspects of these experiments need to be identified. A comparative analysis of the PDFs defined for the studied experiments allowed to identify the most common aspects of scientific experiments:

- **Scientist** performing the experiment (owner of the experiment)
- **Input** (e.g. data acquired from a device)
- A set of **activities** (e.g. computational processes), applied on the input
- **Output** (e.g. data obtained by applying a process on some input data)
- **Devices** (hardware) (e.g. to acquire data, or to perform a process)
- **Software** (e.g. to control a specific device, or to perform a specific process)
- **Conditions** and **parameters** for the processes, devices and software
- A **recursive flow of processes and data** where a specific order is followed during an experiment

The main importance of identifying the common aspects of scientific experiments is that it enables the provision of common (i.e. generic) solutions for supporting scientific experiments, and hence the reusability of these solutions.

**Generic Data Model for the Common Aspects of Experiments**

One major obstacle that most scientists face when performing an experiment is the lack of a generic model for the representation of experiment-related information. The absence of such a generic model does not only cause the scientific database developers to reinvent the wheel every time there is a need to model a new type of experiment, but also hinders the collaboration among scientists due to several different models developed for the same type of experiment.

Therefore, next step in the approach was to develop a data model that contains generic constructs to represent the common aspects of scientific experiments.

A generic data model called *Experimentation Environment Data Model* (EEDM) was developed to uniformly model and represent experiment-related information. The common aspects of scientific experiments form the base for EEDM. EEDM defines generic constructs that correspond to the common aspects of scientific experiments, which are extended (i.e. sub-typed) to model domain-specific experiment-related information. EEDM is described in details in Subsection 4.3.2.
Developing Application Specific Data Models

A PDF includes the experiment components involved in that type of experiments. Therefore, the PDFs can be reused to identify the application-specific experiment components. The next step in the approach was to identify such components, and to develop a data model to represent them by sub-typing the generic EEDM constructs.

The EEDM is flexible and extendible, so that the application-specific (experimentation) data models can easily be developed on top of the EEDM by extending its generic constructs for modelling the application-specific constructs.

Some examples of the microarray specific types that are modelled by extending the generic EEDM constructs are provided in Chapter 6. The microarray application database and its schema are also described in Chapter 6.

Such flexibility and extendibility make the EEDM (and in turn the application database schemas) open to possible changes in the future to support new types of information (that emerge with the advances in the experimental technologies).

4.3.2 Data Modelling for Information about Scientific Experiments

Information about scientific experiments covers experiment procedures, experiment contexts and information about the computational processing involved in the experiments. In this subsection, the data models developed for the modelling of these different types of information about scientific experiments are presented.

Modelling Experiment Procedures: The Procedure Data Model

A procedure is composed of the data types defined in an application database (e.g. microarray database). In other words, every procedure element has a corresponding data type in the application database schema. The Procedure Data Model (Figure 4.13) facilitates the representation and storage of experimental procedures. It allows domain experts to design experiment procedures using the domain-specific application database schemas. The Procedure Data Model is the same for all domains and experiment types, hence all experiment procedures within a CEE can be maintained by a central repository for procedures implementing the Procedure Data Model. Alternatively, the Procedure Data Model can be included in the schemas of the application databases, and every application database maintains its own procedures. The latter allows a clear distinction of procedures based on the domain.

An instance of the Procedure class in Figure 4.13 contains the general information about an experiment procedure. Multiple versions of the same procedure can exist. Every Procedure consists of a set of ProcedureElements. A ProcedureElement corresponds to a data type in an application database (e.g. MicroArray type in the microarray application database), where the corresponding data type for an element is represented by the className attribute. The isProcessing attribute indicates whether the element actually corresponds to a computational activity. The reusePolicy attribute is used to define what to do in case of a query on the type represented by this element. Reuse policy will be described in details in Subsection 4.4.2.
4.3. Data Modelling for Information Related to Scientific Experiments

Following the general structure of experiments, the first two ProcedureElements in an experiment procedure always represent the Project and the Experiment classes. ProcedureElements are connected to each other through ProcedureConnections, which correspond to the relationships between classes, indicated by the relName attribute. ProcedureGUI objects keep graphical information about the procedure, which is used by the Procedure and Context Editors to properly display the experiment procedure.

Modelling Experiment Contexts: The Experimentation Environment Data Model

A context is an instance of a procedure, and the procedure elements correspond to data types in an application database. That means, a context actually consists of a number
of objects in the application database, hence, modelling the application database schema for the domain of an experiment is sufficient for modelling the contexts of that type of experiments.

Using the 'general structure of scientific experiments' and the 'common aspects of scientific experiments', a base data model for experiment-related information called *Experimentation Environment Data Model* (EEDM) is developed (see Figure 4.14). EEDM aims at the storage and retrieval of *generic description of the steps and information* involved in a scientific experiment, for the purpose of *investigation and reproducibility*. EEDM covers the general aspects of an experiment while maintaining its recursive nature. It includes parts from the Dublin Core standard [81].

The *Project* class is the root of the EEDM hierarchy. It keeps information about a project, and groups together the (logically) related experiments. The *Experiment* class lies in the center of the EEDM. It describes the experiment being performed, covering only the high-level information such as the type, subject, and a brief textual description. *Multi-disciplinary experiments* are represented by attaching experiments from different domains under the same project.

EEDM provides generic constructs for modelling experiment components (i.e. activities, data elements, and physical entities). Note that in EEDM, experiment components are collectively modelled as *Steps*. In specific, activities are modelled as *ActivitySteps*, while data elements and physical entities are together modelled as *DataSteps*. Thus, in the remaining of this thesis, the terms *experiment component* and *experiment step* will be used interchangeably.

The most important aspect of the EEDM is that it allows any ordering of experiment components, which enables the model to represent any process-data flow in an experiment. The key to this aspect is the application of recursion as described in Subsection 4.1.1, used within the experiment component (step) definition. The order is given by using recursive *next* and *previous* pointers from one step to another. The same method is used for experiments to represent the order of experiments within a project. An experiment step can also be represented as an aggregation of a number of steps, using the *sub-step* and *super-step* relationships. This allows the abstraction of experiment steps at any desired level. Other experimental information, such as conditions and results are modelled by sub-typing the activity or data steps. Conditions can also be modelled as *properties* attached to a step. *Properties* allow for the representation of semi-structured and ad-hoc information using name-value pairs. Moreover, people can provide *Comments* on experiments or on individual steps.

*Hardware* and *Software* classes provide information about available devices and software programs, and their required *Parameters*. *HwTool* and *SwTool* classes, on the other hand, are used to provide information about the devices and software programs used during experiments with the actual *Parameters* and their values. Furthermore, *User* information is stored for each *Experiment* (as the owner and/or contributors of the experiment) and for each *Step* (as the person performing the activities or owning the data elements and physical entities).

A detailed description of EEDM constructs can be found in [164].
Modelling Experiment’s Computational Processing

A context provides descriptions of processes involved in an experiment, while a computational processing of an experiment provides both static information (i.e. de-
Modelling information about software entities. In order to model the descriptions of software entities, a data model is developed as shown in Figure 4.15. Software entities are self-contained executable programs. They are mainly characterized by the tasks that they perform, their input and output data types, and run-time requirements. A software entity contains the necessary information for a user to choose and use it in her/his processing, such as the description of the functionality provided by this software entity, date that it is registered to the CEE, version number, manual (i.e. documentation for both users and other developers), and run-time requirements for the corresponding executable. The run-time requirements include CPU, memory, and storage requirements. Every software entity has a number of input and output ports that allow a software entity to be attached to other software entities. Ports receive and send data of fixed data types. A software entity may have a number of parameters. A software entity can be atomic representing an executable program, or aggregate consisting of a number of other software entities connected to each other. Every atomic software entity has one or more corresponding executables for each of the platforms that the software entity is implemented, which may accept command line arguments during initialization or depend on certain environment variables for execution. For an atomic software entity, its developer specifies the run-time requirements to help the users to decide which software entities to use in their processing. Additionally, for each software entity, the developer (i.e. a user) specifies the graphical properties, such as an icon representing the software entity to indicate how the Processing Editor should display the software entity.

Modelling computational processing of an experiment. A user defines an experiment’s computational processing by attaching a number of software entities to each other. This computational processing of the experiment must be stored for future references to the experiment and for possible re-runs. For this purpose, the data model given in Figure 4.16 is designed. At run-time, the processes in the computational processing definition of an experiment are connected to each other through their input and output ports. Run-time information associated with each process is also saved, such as the Globus context, location of the log file, the actual host that the process was scheduled for execution, the job description in RSL (Resource Specification Language), and the actual values for the command line arguments. Similarly, the actual values of the parameters and the environment variables used during the execution of a process are stored together with the process. Furthermore, information about the user defining the computational processing for her/his experiment, as well as the graphical information of this definition (to be used by the Processing Editor) are also stored.

Note that an experiment’s computational processing is executable, and every process in it has different information for each execution of the processing. Hence, the
information management platform must distinguish the description of a software entity from the description of a process that is executed as part of a computational processing. In fact, the data model for software entities (Figure 4.15) is a ‘meta-model’ in comparison to the model of processes depicted in Figure 4.16. As such, to distinguish between these two data models, all types in Figure 4.15 are denoted by the prefix ‘m’. Similarly, mParameter, mEnvVariable, mPort in Figure 4.15 are
considered as *meta classes* with corresponding *instance classes* in Figure 4.16.

All relationships between the software entities (meta classes) and processes (instance classes) are depicted in Figure 4.17. Please note that although all classes in 4.15 are considered as meta classes and contain the prefix 'm', only four of them have corresponding instance classes (as shown in Figure 4.17).

### 4.3.3 Data Modelling for Information about Users

It is necessary for a CEE to store and maintain information about its users. This information is used for security and access control (i.e. authentication and authorization) and for administration purposes. In addition, application databases contain user information to capture, for instance, the owner of a physical entity or data element (e.g. microarray or array image), or operator of an instrument (e.g. array scanner). Hence, the information management platform of the CEE must provide the necessary data models for modelling information about users.
4.3. Data Modelling for Information Related to Scientific Experiments

Modelling User Information

The data model developed for user information is shown in Figure 4.18. Username attribute of the User class 'uniquely' identifies a user within the information management platform of a CEE. Value of the certSubject attribute is used for executing a computational processing defined by this user on the Grid. Every user is assumed to have an affiliation with an organization. At least one ContactInfo object is kept for each User and Organization instance.

As mentioned above, user information is included in information about experiment components as well as in administrative information for authentication and authorization, thus it is defined in the data models developed for these types of information. As such, information about CEE users is defined and stored in all databases of the CEE. In application databases, for instance, user information is used to represent the operators of instrumentations or owners of data elements. Information about users can be duplicated in each of the CEE databases. Alternatively, a better approach would be to maintain user information within one database, with cross-database links from objects in other databases to user objects in this database.

Modelling User Roles and Access Rights

In a CEE, every user assumes a role. Roles are used to group users and to define and enforce access rights for each user group. A user may assume only one role at any time. The Application User role is assumed by all users of an application domain.
Figure 4.18: Modelling roles and access rights as part of user information

For instance, all users that can perform microarray experiments within the CEE may have a role called ‘microarray user’. A user can be associated with another role, in which case the ‘Application User’ role is revoked.

Access to data/information in the information management platform of a CEE is controlled through a set of restrictions. It is considered that by default every user has all privileges on all available data/information. Depending on the role that a user assumes, these unlimited privileges are restricted by a set of ‘restrictions’.

The data model for user information is extended to include roles and access rights (Figure 4.18). Although a user can assume one role at any moment, the same role may be assumed by a number of users. A role is identified by its name and the ID associated to it. Several restrictions can be defined for a role, and similarly the same restriction can be defined for several roles. A restriction for a user role is defined for certain operations on a certain set of instances of a certain data type in a certain data source. Following is a brief description of the attributes of the Restriction class:

- **dataType**: This restriction is defined for the instances of the data type specified by this attribute, e.g. experiment procedures or data elements.

- **restrictions**: If this restriction is not specified for all instances of the dataType but for only a subset of them, then this attribute specifies the subset.
For example, a valid “where clause” to be appended to a query statement can be set as the value of this attribute to define a restriction.

- **restrictedOps**: Value of this attribute represents the operations that are not allowed on the instances of the `dataType`. Possible values include `read`, `write`, `read/write`.

- **dataSourceId**: The information management platform of a CEE may manage several data sources. The value of this attribute specifies the data source that holds the data type for this restriction.

### 4.3.4 Data Modelling for CEE-Related Information

During its operation, a CEE itself manages and manipulates certain types of information, such as information about active sessions and information about the services/functionality it provides. This subsection presents the data models developed for modelling and storing such information.

#### Modelling Session Information

Session management is an important functionality in a CEE for controlling and coordinating user activities. A session starts when a user logs-in to the system and stays active until the user explicitly terminates the session. All user activities occur within a session. Within a collaborative session, there is one active user while there can be several observers. In a session, users can manage an experiment procedure, an experiment context, or define and/or execute an experiment’s computational processing.

Although session management is not directly related to information management, for recovery reasons, persistence is required for session information. Furthermore, the information management platform itself requires information about active sessions, for instance, to manage connections to data sources.

A `Session` class is defined to store the session information. The class contains the unique session ID, ID of the user who created this session, ID of the user who is currently active in the session, IDs of the users who are involved in the session as observers (in case of a collaborative session), ID of the experiment context being manipulated in the session and IDs of its procedure and computational processing.

#### Modelling Information about CEE Services

Following the experiment model introduced earlier in this chapter, main CEE services are centered around the management and manipulation of the components of the experiment model; that is, management and manipulation of experiment procedures and experiment contexts, each of which is associated with an active session. Definition and execution of an experiment’s computational processing are also considered as CEE services. For instance, availability of a ‘microarray experiment procedure’ allows a CEE to provide ‘making microarray experiments’ as a service to its users. Thus, the services that are made available by the CEE to its users are highly dependent on
the existing experiment-related information in the CEE. In order to provide a list of services to users, the CEE makes use of the experiment-related information stored and managed by the CEE information management platform.

The list of available services contains information about the types of experiments supported by the CEE (i.e. available experiment procedures), existing experiment contexts and the projects grouping these contexts, and information about the sessions that are active at the time when the list of available services is requested by the user (e.g. at log-in time). All this information is stored in different ways by the information management platform. In order to generate a list of available services by extracting the necessary information from what is available in the information management platform, a number of info classes have been defined, namely ProjectInfo, ContextInfo, ProcedureInfo, and SessionInfo. These classes are used to provide information to the users about the existing projects, experiment contexts, experiment procedures, and active sessions. Note that the list of available services includes instances of these classes, which are created on the fly when the list is requested by a user.

### 4.4 Functionality Modelling for Managing Information Related to Scientific Experiments

Functionality provided by the information management platform of a CEE must satisfy the functionality requirements identified in Subsection 2.6.1. Based on the data modelling presented in the previous section, this section describes functionality modelling for managing information related to scientific experiments. In specific, following subsections describe functionality as well as policies for managing information about scientific experiments (i.e. for managing experiment procedures, experiment contexts and computational processing of experiments), and functionality for managing scientific data/information.

#### 4.4.1 Functionality for Managing Information about Scientific Experiments

Functionality provided by the CEE information management platform for managing information about scientific experiments includes management of experiment procedures, experiment contexts, and computational processing of experiments.

**Managing Experiment Procedures**

Information management platform of a CEE provides the base experiment procedure management functionality, such as creating a new procedure, reading and updating an existing procedure, saving an existing procedure as a new procedure, and deleting a procedure. Scientists can only read and instantiate experiment procedures, while domain experts can also manipulate them. A scientist reads an experiment procedure using the Context Editor, either when creating a new experiment context using that
procedure or when loading an existing experiment context that was created by using that procedure. Domain experts manipulate experiment procedures using the Procedure Editor when defining a new procedure or when working on an existing procedure. An experiment procedure is managed as an atomic unit in the CEE information management platform, meaning that the entire procedure is read or manipulated in a single operation.

An experiment procedure is stored in the application database that is specifically developed for storing the information related to experiments from the domain of the experiment procedure. An application database also contains the experiment contexts that are instances of the experiment procedures stored in that database. When creating a new experiment procedure, a domain expert must specify the database for which s/he is going to create the procedure, so that the information management platform can retrieve the data types defined in the database schema and store the newly created experiment procedure in that application database. Since the elements of a procedure correspond to the data types in the schema of the application database, these data types can be retrieved from the database and presented to domain experts within the Procedure Editor.

Using the Procedure Editor, a domain expert can design a new experiment procedure. When designing a procedure, the domain expert neither has to use all of the types in the database schema, nor has to use all of the relationships in a type. Depending on the type of the experiment for which the procedure is being defined, irrelevant relationships can be filtered out by excluding any ProcedureConnections in the experiment procedure that correspond to these relationships. In other words, a ProcedureElement does not necessarily have to define a ProcedureConnection for each one of the relationships defined in its corresponding data type. Figure 4.19 explains this filtering out of relationships using an example from microarray experiments. Left side of the figure shows the MicroArray element of the experiment procedure for microarray production and its related generic procedure elements. In the procedure, the ProcedureElement corresponding to ArrayTemplate has one outgoing ProcedureConnection, which corresponds to the usedByArray relationship (not shown in the procedure). The UML model for the ArrayTemplate type is given on the right side of the figure. The ArrayTemplate type contains the relationship hasSpots in addition to the relationship that is included in the procedure element (i.e. usedByArrays. The hasSpots relationship is filtered out from the experiment procedure because there are thousands of Spot objects for a single ArrayTemplate, which can not be displayed within the Context Editor when instantiating this procedure.

The experiment procedure management functionality supports version management. Multiple versions of the same procedure can co-exist, where there may be one or more experiment contexts created by instantiating a version of the procedure. Version management for experiment procedures is described in details later in this subsection.

When creating a new experiment procedure, value of the procedureGroupId attribute is set to the value of the id attribute, indicating that a new version group for this experiment procedure is created. The procedureGroupId attribute is used
to group together the procedures that represent the same type of experiments but have different versions. In other words, different versions of the same procedure are grouped together by assigning the same value to their procedureGroupId attributes.

When reading an experiment procedure, a domain expert must specify the target data resource and the ID of the experiment procedure. On the other hand, when a scientist reads an experiment context, its corresponding procedure is implicitly retrieved along with the context itself. The id attribute of the Procedure class is used to uniquely identify the entire procedure. Upon receiving this information, the information management platform reads the Procedure object with the specified ID, and continues reading all ProcedureElement, ProcedureConnection, and ProcedureGUI objects defined as part of this experiment procedure. The oid attributes of the objects in the procedure contain positive values assigned by the data resource *. Older versions of an experiment procedure can be retrieved by specifying its procedure ID. This is useful if a domain expert wants to use an older (version of a) procedure as the starting point when designing a new one.

Updating an experiment procedure is similar to creating a new one, though it

*In the CEE information management platform, a positive OID for an object means that the object already exists in the specified database, while a negative OID means that the object is new and needs to be inserted into the specified database
involves several more steps. A domain expert reads an existing experiment procedure as described, and modifies the procedure by changing attribute values or by adding/removing procedure elements. When the request to update an existing experiment procedure arrives, the information management platform first checks whether there are any experiment contexts in the data resource created by instantiating this experiment procedure. If there are no such experiment contexts, then the existing procedure is removed from the data resource, and the updated experiment procedure is re-inserted. The removal of the old procedure is necessary, because, if the updated procedure contains less number of ProcedureElements than the one already in the data resource, the only way for the information management platform to determine the deleted ProcedureElements is to perform a one-to-one comparison of the elements in the new procedure against the ones in the old procedure. To avoid the inefficiency associated with this comparison, the existing procedure is removed first. Before inserting the new procedure, all positive oid values in the experiment procedure are replaced by negative values to ensure that new objects are created in the data resource. During this replace operation, the information management platform makes sure that the consistency among the OIDs is preserved (i.e. the OID values used within relationships are also replaced with the correct negative values). On the other hand, if there are instances of the existing procedure, then the information management platform inserts the updated experiment procedure as a new one, without removing the existing procedure. Removing the procedure of an experiment context would lead to inconsistencies, both because it breaks the policy of ‘no experiment context can exist without an experiment procedure’, and because the experiment procedure contains additional information for the representation of the context on screen. To avoid such inconsistencies, a versioning system is designed for procedure management.

When creating a new version of an existing experiment procedure, the information management platform checks the latest version number in the version group of the procedure (recall that different versions of an experiment procedure have the same procedure group ID), increments this version number by one, and assigns it to the new experiment procedure. Thus, the newest experiment procedure has always the largest version number. After the experiment procedure is successfully saved, it is returned back to the user with the updated OID values.

Experiment procedures can be deleted by specifying the ID of the procedure to delete. Similar to updating an experiment procedure, if there are no experiment contexts created by instantiating this procedure, the procedure is physically removed from the data resource. Otherwise, the experiment procedure is not physically removed, but it becomes invisible to users. Then on, although it can be retrieved together with one of its corresponding experiment contexts, no new contexts can be created using this experiment procedure.

It is also possible to save an updated experiment procedure as a new one. In this case, all OIDs in the experiment procedure are negated, and the request is treated as inserting a new experiment procedure.
Managing Experiment Contexts

The information management platform supports management of experiment contexts within a CEE. The base experiment context management functionality includes creating a new context, reading, updating and deleting an existing context, and saving an existing context as a new one. An experiment context is managed as an atomic unit in the CEE information management platform.

For a complete description of the experiment context management functionality, some key concepts need to be introduced, related to the link between an experiment context and its corresponding procedure, and the links between an experiment context and any computational processing defined as part of this context. Object sharing and reusing will be described in the following section, which is another essential concept for managing experiment contexts.

Context-procedure links. As described earlier, an experiment procedure contains information about the graphical representation of the experiment contexts that are created using this procedure. Thus, for proper presentation of an experiment context, a link between the experiment context and the experiment procedure must be created and maintained during the life time of the context. For this purpose, two classes are defined, namely ContextProcedure and ContextElmProcedureElm, which are included in the application database schemas.

One ContextProcedure object is created for each experiment context in an application database. It indicates which experiment procedure is used to create the context. A procedure can contain more than one procedure elements that correspond to the same data type in the application database schema (e.g. procedure elements for ‘operators’ and ‘owners’ that both correspond to ‘User’ type in the schema), which makes it impossible for the Context Editor to find the correct experiment procedure element for a given object in the context only based on the type of the object. Hence, for every object in an experiment context, a ContextElmProcedureElm object is created to establish the link between the object in the context and the procedure element that was used to create the object.

Context-computational processing links. The experiment model described in Section 4.1 requires that every computational processing is defined as part of an experiment context, since processes in a computational processing correspond to experiment components in an experiment context. In other words, the description (i.e. context) of a computational processing must already exist before the computational processing is executed. Similar to the link between an experiment context and its corresponding experiment procedure, links must be created and maintained between an experiment context and any computational processing within this context. In order to relate a computational processing defined by a user to the context element corresponding to this processing, a pointer is defined in that context element. As shown in Figure 4.13, every ProcedureElement contains a flag indicating whether the ProcedureElement corresponds to a computational processing. For the ProcedureElements that have this flag set to true, the corresponding data types must define an attribute called
processingId, to serve as a pointer to the computational processing represented by an instance of this data type.

**Context management.** In order to create a new experiment context, the procedure for the experiment must be already defined by a domain expert. After selecting the procedure for the type of the experiment that s/he wishes to perform, the user walks through the elements of the procedure one by one, and creates the new experiment context either by creating new objects for the elements or by issuing a query to retrieve (a copy of) an existing object. The user also links the newly created objects to each other. Context Editor guides the user through the context creation.

When creating a new experiment context, the necessary context-procedure link objects and origin-copy objects are also stored together with the experiment context (origin-copy objects will be described in Subsection 4.4.2). Similar to objects in an experiment procedure, a new experiment context contains several objects (i.e. context elements), which are either newly created objects with negative OID(s) or existing objects with positive OID(s) (e.g. an already existing object representing the operator of an instrument). New objects are created in the application database, while existing objects are updated if the user has the required (update) privileges. Otherwise, existing objects remain unmodified, except that the relationships between existing and new objects are created.

In order to read an existing experiment context, the user must specify its ID. Context ID uniquely identifies an experiment context within the scope of an application domain. The information management platform uses the specified ID and the context-procedure links to read the objects that belong to the requested experiment context.

In an updated experiment context, some of the objects may be updated and some of them may be newly created, while the other objects may remain unmodified. Unlike the experiment procedure management, no versioning mechanism is used for experiment context updates; all existing data is overwritten with new data. Furthermore, since it is not possible for the information management platform to determine which objects have actually been updated, all objects are re-written regardless of whether they have been updated or not. Updates also do not allow users to delete objects from an experiment context, since there are no mechanisms for determining the deleted objects. The consequence otherwise would be that the objects deleted from an experiment context are not physically removed from the database, but disconnected from the experiment context, which would cause both dangling objects in the database and inconsistencies with the context-procedure links and with query processing. Updating an experiment context also updates the context-procedure links.

Deleting an experiment context considers the reuse policy defined on the procedure elements. When deleting an experiment context, the information management platform determines which objects should actually be deleted and which should be kept in the database based on the reuse policy values. Only the objects whose corresponding ProcedureElements have CREATE_COPY as reuse policy value will be physically removed from the database. Deleting an experiment context also deletes the context-procedure links.
When saving an existing experiment context as a new one, some of the objects in that context are included also in the new one, while a copy must be created for some other objects. The same algorithm used for determining the objects to be deleted is also used for determining the objects to be copied. All objects for which the corresponding ProcedureElements have CREATE.COPY as reuse policy value will be copied, while the remaining objects will be directly used in the new experiment context. Please refer to Subsection 4.4.2 for a detailed description of reuse policy.

Managing Experiment’s Computational Processing

Provided functionality for managing an experiment’s computational processing also includes the functionality for managing information about software entities.

Managing information about software entities. The CEE information management platform supports the base functionality for creating, reading, updating and deleting descriptions of software entities.

Software entities are developed by tool developers, but they are registered to the CEE and their descriptions are stored in the information management platform by domain experts. In order to store the description of a software entity, a domain expert provides all necessary information shown in Figure 4.15. Information about a software entity is managed as an atomic unit in the CEE information management platform.

When a request arrives to read the description of a software entity, the information management platform retrieves the description with the specified ID together with the related objects (see Figure 4.15) and returns them to the requester.

Updates to software entity descriptions are handled similar to experiment context updates. All existing information is overwritten with the updated information provided by the user.

When a request arrives for deleting a description, all its related objects are also removed from the database, except the developer information.

Saving an existing software entity description as a new one is also supported. This functionality can be used when, for instance, a new version of the software entity is being registered.

Managing computational processing of an experiment. The CEE information management platform provides the functionality for creating a new computational processing, reading, updating and deleting an existing computational processing, and saving an existing computational processing as a new one.

As described earlier, a computational processing for an experiment is defined by attaching a number of software entities to each other. The computational processing is stored in the information management platform using the data model given in Figure 4.16.

Since the software entity descriptions can only be created/updated by domain experts, a computational processing does not contain any software entity descriptions itself, however, processes in the computational processing are linked to the descriptions of their corresponding software entities. Hence, only a single copy of a software
entity description is saved in the database, which can be used to describe several processes. A computational processing is defined using the Processing Editor. When a computational processing is completely defined, all Process objects in the processing are stored in the database before execution. A computational processing is treated as an atomic unit by the CEE information management platform.

Similar to creating a computational processing for an experiment, reading a computational processing returns only the Process objects with links to software entity descriptions. Descriptions of software entities are assumed to be retrieved in a separate request.

When updating an existing computational processing, first the existing computational processing is removed from the database, then the updated one is inserted.

Deleting a computational processing of an experiment removes all objects in the processing from the database, except the user information. Links to the software entity descriptions are also removed.

The information management platform supports saving an existing computational processing as a new one, for instance, in case of a parameter change.

Recall here that when a user defines a computational processing using the Processing Editor, a pointer to this processing is created in the experiment context that includes this processing. The pointer consists of the processing ID. However, since updates to a computational processing first remove the existing one and then insert the updated one, under normal conditions the computational processing obtains a new ID. To prevent any inconsistencies after update or save as requests, the information management platform makes sure that the ID of the ComputationalProcessing object remains the same during the lifetime of the object. Thus, the computational processing can be retrieved using the same ID during its lifetime.

4.4.2 Policies for Managing Experiment Contexts

Based on the given conditions, a policy defines a method of action selected from a set of alternatives. In a complex collaborative environment like a CEE, users share information at different levels. Two kinds of policies are defined and used for the management of experiment contexts: Reuse policy and update policy. Reuse policy is used to define the rules for sharing objects, such as defining the set of objects that are related to a shared object and hence that also need to be shared, or how to handle sharing of information about instruments or software for which only a single object exists. Update policy defines what to do in case when a shared object is updated. Below, first the issues that need to be considered about sharing and reusing objects are introduced. Then the two types of policies are presented, and how the reuse policy is used during experiment context management is described.

Sharing and Reusing Objects

Users may need to share experiment-related data/information when working on an experiment context. Information to share can be either at the level of an entire experiment context or at the level of objects in a context. For example, consider the
case where a user wants to perform an analysis experiment on some sample, which was prepared in another experiment in another project. In order to use the (description of the) sample in her/his experiment, the user must first copy the sample object from the original experiment context to her/his context, which is under a different project than the original context. Once the sample object is copied into the new experiment context, the user can continue working on it; for instance, by using the sample as input to an FTIR measurement process as the next step in the experiment.

The OriginCopy type is defined for maintaining information about the origin and destination of such shared objects. When an object is copied from another experiment context, an instance of this type is created, which contains the OID of the origin object and the OID of the copy object. This object is used to assist users with tracing the original copy of an object, and to maintain the consistency in case of updates to the original copies (which will be described later).

Sharing of entire experiment contexts is explicit, and supported through saving the original experiment context as a new context, possibly as part of a project other than the origin project. Sharing of single objects, however, is performed implicitly when the user issues a query on an experiment component (represented as a context element). Query functionality will be described later in this section.

In some cases, only a single copy of an object can exist (i.e. a singleton object). This is usually the case when the objects are created by domain experts or administrators and can not be updated by ordinary users. For instance, only a single copy of a User object can exist in a CEE at any time; hence, a user object can not be copied, but it must be reused. That is, when working on an experiment context, instead of creating a copy of the user object and including the copy in the context, objects in the context are directly linked to the original user object through relationships. This way, only a single copy of the object is preserved, while that object is reused by many users in different experiments. Other examples of such singleton objects include objects representing an instrument or a software program that are used by all users (i.e. hardware/software objects), or domain-specific objects that are maintained by domain experts and can not be updated by users, such as an object representing a gene or an organism.

Another important issue on object sharing/reusing is related to the semantic relationships between the shared/reused object and other objects, which at the same time define the semantic object boundaries for the shared/reused object. To better illustrate this issue, consider the example where a user wants to spot new microarrays using the same settings as the array spotting process that s/he performed last time. A simplified version of the procedure for this experiment is shown in Figure 4.20. In order to make this new array spotting with the same settings as the last experiment, the user has to copy the ArraySpotting object from the context of that experiment. However, although the ArraySpotting object contains information about the details of this process, this object itself is not sufficient for using the same settings. Other objects which are semantically related to ArraySpotting (i.e. the hardware/software tools and the protocol used to make the process, extra information provided as properties, and -if any- even the comments of other users) must be copied as well. These semantically related objects are encircled with dashed lines in Figure 4.20.
Figure 4.20: Semantically related objects (semantic object boundaries) in an experiment procedure
The Reuse Policy

In the information management platform, reuse information and semantic relationships are encoded by domain experts as policies. Since reuse information and semantic relationships are common to all experiments of the same type, this information can be attached to the corresponding experiment procedure. The `reusePolicy` attribute defined in the `ProcedureElement` and `ProcedureConnection` types of the Procedure Data Model (shown in Figure 4.13) is used for this purpose. When defining a new procedure, a domain expert sets the reuse policy for each `ProcedureElement` and `ProcedureConnection`.

Reuse policy is in general used for evaluating queries, for finding objects that are semantically related to a given object, and for deleting experiment contexts. Meaning of the `reusePolicy` attribute value differs depending on where it is defined, i.e. in a `ProcedureElement` or in a `ProcedureConnection`.

The reuse policy values for `ProcedureElements` refer to the policy to be applied to retrieve the instances of the data type corresponding to a `ProcedureElement` when a query is issued on that `ProcedureElement`. After issuing the query, the user selects one object from the query result set (or more depending on the number of objects returned and on the cardinality specified for the `ProcedureElement`). The reuse policy states whether the object(s) selected by the user should be copied or reused. The possible values for Reuse Policy are shown in Figure 4.21.

As mentioned earlier, meaning of a reuse policy value is different for `ProcedureElements` than for `ProcedureConnections`. Below, meanings of possible reuse policy values for `ProcedureElements` when processing a query are briefly described.

**REUSE_POLICY_NOREUSE.** If the reuse policy is set to NOREUSE, no objects are returned back as the query result. In other words, NOREUSE means that the data type corresponding to this `ProcedureElement` can not be queried.

**REUSE_POLICY_REUSEORIGINAL.** In this case, the original object in the database (which is selected by the user from the query result set) is returned back to the user.

**REUSE_POLICY_CREATECOPY.** If the reuse policy is set to CREATE_COPY, then the original object in the database is not directly returned to the user but a

```
REUSE_POLICY_NOREUSE
REUSE_POLICY_CREATE_COPY
REUSE_POLICY_REUSEORIGINAL
REUSE_POLICY_CREATE_COPY_AND_LINK
REUSE_POLICY_REUSEORIGINAL_AND_LINK
REUSE_POLICY_REUSEORIGINAL_AND_NOLINK
```

Figure 4.21: Possible values for the Reuse Policy
copy of that object is created and returned back when the user selects an object from the query result set. In this case, one OriginCopy object must be created for each copied object. Here, only the attributes of the object are copied (not the relationships).

Reuse policy values on ProcedureConnections refer to the objects that are semantically related to a given starting object (i.e. the object selected by the user from the query result set). After determining what to do with the starting object by applying the reuse policy on the queried ProcedureElement, other objects that are related to this starting object are identified. Whether these objects should be copied or reused is determined based on the reuse policies defined on the ProcedureConnections.

The possible reuse policy values for ProcedureConnections have the following meanings for querying:

**REUSE POLICY NOREUSE.** If a ProcedureConnection has NOREUSE as the reuse policy value, that connection is not followed during query processing, that is, the relationship represented by this ProcedureConnection is not traversed and its successor(s) are not retrieved from the database.

**REUSE POLICY REUSE ORIGINAL AND LINK.** In this case, the successor of the relationship represented by this ProcedureConnection is read from the database, and a link from the starting object to this object is created (i.e. the starting object and this object are related to each other). This (original) object is returned back to the user.

**REUSE POLICY REUSE ORIGINAL AND NOLINK.** This case is similar to the previous one, except that no link is created between the starting object and the successor of the relationship. The successor object is returned back to the user. This policy value is used to prevent double-links when the starting object and the successor of the relationship are already related in the database.

**REUSE POLICY CREATE COPY AND LINK.** A copy of the successor of the relationship is created and linked to the starting object. In addition, one OriginCopy object is created for each copied object. Again, only the attributes of the successor object are copied (not the relationships).

As described earlier, in some cases only a single copy of an object can exist. When deleting an experiment context, special attention is needed if such an object is used in the context. All objects in the experiment context must be physically removed from the database except this object. The reuse policy defined for a ProcedureElement is also used to determine whether the object(s) in a context corresponding to this ProcedureElement should be deleted or not. The reuse policy values for ProcedureElements have the following meanings for deletion:

**REUSE POLICY NOREUSE / REUSE POLICY REUSE ORIGINAL.** When deleting an experiment context, the objects whose corresponding ProcedureElements have these values as reuse policy are not deleted.
**REUSE_POLICY_CREATE_COPY.** All objects in the experiment context with CREATE_COPY as the reuse policy value are physically removed from the database.

**The Update Policy**

Copying an object may introduce consistency problems. Since the origin of the object is another experiment context owned by another user, that user may update or delete her/his experiment context any time. Hence, the information management platform must support the definition and enforcement of *Update Policies* for the objects that are copied. The update policy is defined as an attribute in the *OriginCopy* class. The Update Policy is defined by a user for the objects copied by her/him. Possible values for Update Policy are shown in Figure 4.22.

When an origin object is updated, all its copies are checked using the *OriginCopy* instances. Depending on the update policy defined for the copy object, the proper action is taken. Below is a brief description of each possible value for the update policy:

**UPDATE_POLICY_NOUPDATE.** If the user sets the update policy value to NOUPDATE for the copied object, no action is taken when the origin object is updated.

**UPDATE_POLICY_AUTO_UPDATE.** If the update policy value is set to AUTO_UPDATE, the changes are reflected to the copy object when the origin object is updated by propagating the new state of the origin object to the copy object. Here, only the attribute values of the copy object are updated, while its relationships are kept untouched.

**UPDATE_POLICY_NOTIFY.** If the update policy value is set to NOTIFY, then a notification message is sent to the owner of the copy object when an update is requested to the origin object. Upon receiving the notification, the owner of the copy object must decide whether to reflect the changes or to ignore them.

**Applying the Reuse Policy**

In Figure 4.20, the semantically related objects for a starting object in an experiment context were depicted. Figure 4.23, on the other hand, emphasizes the reuse policies defined for the related objects and the relationships, and is used for illustrating how

![Figure 4.22: Possible values for the Update Policy](image-url)
reuse policies are applied. The experiment model plays an important role in the enforcement of reuse policies.

Recall the example where a user wanted to spot new microarrays using the same settings as the array spotting process that s/he performed last time. For this purpose, s/he issues a query to retrieve the description of the last array spotting process that s/he performed. The result set will contain only one ArraySpotting object (i.e. the one with the latest timestamp). Operations involved in the processing of this query are enumerated below:

1. When s/he selects this object from the query result set, the experiment context that contains the selected ArraySpotting object and the experiment procedure used to create this context is located by the information management platform. The selected ArraySpotting object will be here on referred to as the starting context element, and the ProcedureElement corresponding to the starting context element will be referred to as the starting procedure element.

2. The information management platform checks the reuse policy defined for the starting procedure element, which contains CREATE_COPY as the value. This means that the user will not actually use the object that s/he selected from the query result set (i.e. the starting context element), but a copy of it. The information management platform creates a new ArraySpotting object, copies the state of the existing object to this one, and inserts the copy object to the return list (i.e. the query result set).

   If the reuse policy for the starting procedure element had REUSE_POLICY_NOREUSE as value, then an empty return list would be returned to the user; or if it had REUSE_POLICY_REUSEORIGINAL as value, then the starting context element itself would be inserted into the return list.

The next step is to find out the objects that are semantically related to the copied ArraySpotting object. To determine which objects are related, the link between an experiment context and its procedure is used. Starting from the starting context element, the information management platform recursively navigates through the relationships of the starting context element. In parallel, starting from the starting procedure element, the information management platform recursively traverses its connections, while for each step considering the ‘reuse policy defined for the connection’. Depending on the reuse policy value, either a copy of the traversed object in the experiment context or the object itself is inserted into the return list. All relationships/connections are recursively traversed until either all the connections are traversed or all the remaining connections have NOREUSE as the reuse policy value.

3. In the example of Figure 4.23, the starting procedure element (which corresponds to ArraySpotting) has seven outgoing connections. Assume that the first ProcedureConnection to traverse is the one corresponding to the relationship usesHWTool. The information management platform traverses this ProcedureConnection and stops at the ProcedureElement corresponding to AS Hw Tool. In parallel, the information management platform navigates through
Figure 4.23: Example of reuse policies
the corresponding relationship in the experiment context, and stops at the **HwTool** object.

Because the reuse policy value for the traversed **ProcedureConnection** is CREATE_COPY_AND_LINK, a copy of the **HwTool** object is inserted into the return list and it is linked to the copy of the starting **ArraySpotting** object, which was already inserted into the return list.

4. Similarly, continuing from the **HwTool**, the first **ProcedureConnection** and its corresponding relationship (hasHardware) are traversed.

Because the reuse policy value for the **ProcedureConnection** is REUSE_ORIGINAL_AND_LINK, the actual **Hardware** object (representing the array spotter device) is inserted into the return list and linked to the copy of the **HwTool** object (which was already in the return list).

5. The only outgoing **ProcedureConnection** (from the Array Spotter procedure element) and its corresponding relationship (hasParameters) are traversed, and the actual **Parameter** object is inserted into the return list. Note here that the **Parameter** object is not linked to the **Hardware** object, because both objects are the actual objects in the database, and there is already a relationship between these two objects. That is why the reuse policy value REUSE_ORIGINAL_AND_NOLINK is used for this case.

6. At this stage, because the **ProcedureElement** for the **Parameter** class does not contain any outgoing **ProcedureConnections**, the recursive algorithm rolls back until the **ProcedureElement** for **HwTool**, and continues with the remaining **ProcedureConnection** corresponding to the relationship hasParameters.

In the same way, all outgoing **ProcedureConnections** from the **ProcedureElement** corresponding to **ArraySpotting** are traversed. Note that the **ProcedureConnections** for the relationships isPartOfExperiment and hasNextStep are not traversed. The final return list contains copies of **ArraySpotting**, **HwTool**, **Parameter** (representing the actual parameters for the hardware tool), **SwTool**, **Parameter** (representing the actual parameters for the software tool), **Comment**, **Property**, and **Protocol** objects; the original **Hardware** (representing the array spotter device, **Parameter** (representing the formal parameters for the array spotter device), and the original **Software** (representing the array spotter software, **Parameter** (representing the formal parameters for the array spotter software) objects.

Note that one **OriginCopy** instance is created for each copied object and inserted into the return list, and one **ContextElmProcedureElm** object is inserted into the list for each object returned (existing **ContextElmProcedureElm** objects for reused objects and new ones for copied objects). The list is then returned back to the client side, and the objects in the list are presented to the user within the Context Editor.

### 4.4.3 Functionality for Managing Scientific Data/Information

As mentioned in Section 1.3, existing file manipulation facilities (e.g. those provided by the Grid) are considered in this thesis for the management of very large data sets
that are stored in files. In this direction, large data sets associated with an experiment are not stored as part of the experiment context. The data sets are stored separately as files, and a URL pointing to the file is maintained in the corresponding data element in the experiment context. For example, the ArrayImage data element in microarray experiments contains an attribute called imageLocationURL pointing to the location of the actual image file on the file system.

The data/information management functionality provided by the CEE and its information management platform include generic file manipulation tools, database query functionality, access to multiple databases, and version control.

File read, write, and transfer functionality is provided by the generic CEE software entities. For instance, in order to transfer a file, a user must define a computational processing using two software entities; the first one reads the source file and the second one writes it to the destination.

The information management platform allows its users to issue any kind of queries against a specified database. Users can issue queries in two ways: Either through the Context Editor to query the elements of an experiment context (as described earlier in this chapter), or by using a software entity for querying from within an experiment's computational processing.

Some of the information management functionalities require accessing multiple databases. For instance, since experiment procedures are stored in application databases, in order to provide the users with a list of available experiment procedures, the information management platform accesses all application databases to retrieve the information about procedures from each database, merges the results, and sends the merged result to the user.

Version control is supported in the information management for experiment procedure management. Different versions of the same experiment procedure can co-exist, while users always use the latest version when creating new experiment contexts. A similar versioning mechanism can also be provided for other functionalities for experiment-related information.

### 4.5 Functionality Modelling for Collaboration, Administration and Security

This section focuses on two aspects of modelling collaboration, administration, and security related functionality: Management of information related to collaboration, administration, and security (e.g. managing user information and access rights), and providing the actual functionality for collaboration, administration, and security (e.g. mechanisms for accessing multiple databases, supporting multi-disciplinary projects, and enforcing access rights). Following subsections describe the modelling of collaboration, administration, and security functionality provided by the CEE and its information management platform, also considering the two aspects mentioned above.
4.5.1 Collaboration Functionality

The collaboration functionality provided by the CEE information management platform includes data/information sharing and support for multi-disciplinary projects. Data/information sharing allows users to share experiment-related information at different levels, ranging from a single object to a complete experiment context. Data/information sharing was described in details in Subsection 4.4.2, hence this subsection focuses on functionality for supporting multi-disciplinary projects.

The general structure of experiments (depicted in Figure 4.6 in Subsection 4.1.1) enables the representation of multi-disciplinary projects in the CEE information management platform. The Experimentation Environment Data Model (EEDM) follows this structure and defines generic constructs for modelling scientific experiments (see Subsection 4.3.2). In addition to its important role in modelling different types of experiment-related information, EEDM also provides support for representing multi-disciplinary research. For instance, it supports the representation of inter-related biological, chemical, and physical properties of a certain biological element at once.

Multi-disciplinary research is supported through the project concept. As defined in the general structure of scientific experiments, a project groups together related experiments, and contains higher-level information about these experiments. However, since all experiments also follow the general structure, one can define a project containing experiments from different disciplines. For example, consider a multi-disciplinary project that studies different properties of a certain biological sample, as depicted in Figure 4.24. This project may contain one microarray experiment from biology domain and one material analysis study from physics domain, allowing a scientist to look at the characteristics of the biological sample from two different points of view. Results of these two studies can be further compared with the results obtained from simulations. As such, collaboration among scientists from different domains is supported.

The CEE information management platform allows users to define and manage
multi-disciplinary projects. Experiments in such projects can be from any of the e-science domains supported by the CEE. The provided mechanisms are uniform, that is, the same mechanisms are used for managing both multi-disciplinary and intra-disciplinary projects and experiments.

4.5.2 Administration Functionality

The CEE information management platform provides functionality for supporting both its own administration and the administration of the CEE. The administration functionality includes user management, session information management, and CEE services information management.

User Management

The data model for user information was described in Subsection 4.3.3. User management functionality of the CEE information management platform allows the management and manipulation of user information conforming to this data model. Thus, the functionality covers inserting information for a new user and reading, updating, and deleting existing user information. User information is managed as an atomic unit which also includes the contact information. Since the User type is defined in multiple databases, there can be several User objects corresponding to the same real user. Thus, the information management platform must ensure the consistency and uniqueness of the information about a real user. This can be achieved in several ways; for instance, a central repository can be used to maintain all user information, where other databases provide cross-database links to this repository for user information; or each database maintains a local copy of the user information. In the latter case, mechanisms must be defined to ensure the consistency of the local copies. User management functionality also covers the management of user accounts on the database management systems for providing access to the databases.

In a CEE, it is assumed that every user is an employee of an organization. Thus, the information management platform supports creating, reading, updating, and deleting information about organizations, which also includes the contact information for organizations. In order to insert information about a new user, the organization that s/he is working for must be already defined.

Managing Session Information

As mentioned in Subsection 4.3.4, session information is stored in the CEE information management platform for recovery reasons. Information about an active session is represented as a Session object, which was defined in Subsection 4.3.4. Persistence and management functionality for session information includes persisting it when a new Session object is created in the CEE, updating the persisted Session object when there is a change in the active session which is propagated to the information management platform, and terminating a session by removing the Session object from the information management platform.
Note here that role of the information management platform for session information management is to provide persistence for Session objects. The information management platform is passive in any means, that is, it never modifies any Session object that it persists in a database.

Managing Information about CEE Services

The data types for services information management were described in Subsection 4.3.4, which provide information about existing experiment procedures and contexts, and active sessions. This information is stored and managed by the information management platform. Therefore, the functionality provided by services information management includes extracting the necessary information from different databases, creating the corresponding info objects, and sending them back to the requester.

Note that these objects are not stored in the information management platform persistently; they are created on request by querying the databases to extract the necessary information.

4.5.3 Security Functionality

Users must be authenticated and authorized before they start using any functionality offered by the CEE information management platform. As mentioned earlier, every user has a CEE-wide unique username, which is used for authentication. The CEE may require another level of authentication (e.g. a Grid-based authentication mechanism). However, the information management platform supports single sign-on. Users do not explicitly log-in to the information management platform, rather the information management platform obtains the user information necessary for log-in from the information about active sessions (i.e. from the Session objects). Because most of the DBMSs currently require a username/password for their users, a single password is defined, which is the same for all users. This password is only known to the information management platform and is never made available to users, even to other components of the CEE. When connecting to a database, combination of the username in the Session object and this password is used.

Authorization also takes place at two levels. After a user is authenticated by both the CEE and the information management platform, the information management platform provides a programming interface to the CEE. This interface contains only specific functions that can be called by the subsequent requests of the user, which are selected depending on the role that the user assumes. For instance, any request coming from an administrator will be executed using the administrator interface, which contains the functionality specifically defined for administrators (e.g. functionality for user management).

The second level of authorization is achieved through the access rights management. The access rights management considers the CEE and information management platform users, the roles they assume, available data types (which are stored and serviced by the information management platform), and the access restrictions defined for each role on a given data type.
The data types consist of the different types of information managed by the information management platform. Although it is called as 'data type', a data type can actually consist of several types in a database, such as an 'experiment context data type' which spans over multiple types in the database schema. Thus, the data types include experiment-related data types (i.e. experiment procedures, experiment contexts, and computational processing), software entity definitions, session information, and user information and access rights (i.e. roles and restrictions). Other types of information that are considered as data types include the common password which is not made available to users, and information objects (e.g. a ProjectInfo object) that are not stored in a database but generated on request. These data types are used to define the restrictions that are applied to users when requesting a service that manipulates one of these data types.

As described earlier, every user assumes a role, and the Application User role is the default role assumed by all users of an application domain. Descriptions of the Application User role and the other roles defined for CEE users are as follows:

**Application User.** Application users can only read and modify their own application-specific data (e.g. experiment contexts that they have created). There is one specific application user role for each application domain (e.g. Microarray User, Material Analysis User).

**Administrator.** There are three types of administrator roles:

- **CEE Admin.** In order to achieve uniformity in CEE user management and information management platform user management, user management task is assigned to the CEE Admin role. A CEE Admin is responsible for creating/modifying user accounts both in the CEE and in the information management platform. However, a CEE Admin can not define the access rights and can not read/modify any application-specific data.

- **Information Management Admin.** Information Management Admin is responsible for managing the information about the data sources available within the CEE (e.g. application databases), and defining the access rights on the data types internally used by the information management platform, such as user information and the common password.

- **Domain Expert.** Domain Experts are responsible for creating/maintaining application specific experiment procedures and software entity descriptions. They can also define access rights on these data types, but can not read/modify application data of users. For each domain, one Domain Expert role is defined (e.g. Microarray Expert, Material Analysis Expert).

**System.** This role represents the information management platform. It has all the privileges for all operations. This role is not used during the normal operation of the information management system.

Access to data/information in the CEE is controlled through a set of restrictions. Every user by default has all privileges on all available data/information, which are
restricted by a set of \textit{restrictions} depending on the role that the user assumes. Restrictions are at two levels: Operation level and data level. At operation level, there are three restrictions: \textit{read} (i.e. users can not read a certain data type), \textit{write} (i.e. users can read but can not update a certain data type), or \textit{read-write} (i.e. users can neither read nor update a certain data type). At data level, the value set of a certain data type (i.e. set of instances) is restricted by a number of constraints. These constraints are specified as part of the role definitions.

Figure 4.25 gives the restrictions defined for microarray users as an example. The figure defines the restricted operations for each data type in the microarray data source. Since there can be other data sources for other application domains, restrictions on certain data types (e.g. experiment procedures, experiment contexts) consider the data source holding the actual data/information. For instance, a microarray user can only read the experiment procedures in the microarray data source but can not read or write any other experiment procedure from other data sources (unless this permission is explicitly given by a domain expert).

In some cases, run-time information may be needed to specify restrictions on a certain data type, such as the username of the active user, that is not known at the restriction definition time. For this purpose, parameterized keywords are defined, which are substituted with the actual values at run time. These parameters are in the form of \textit{“$ClassName._AttributeName”}; where class name and attribute name indicate what run-time information needs to be used within this restriction. For example, \textit{“hasOwner.name = $User.name”} specifies that this object can only be read/modified if the name of the active user is the same with the name of the owner of this object (i.e. if the active user is the owner of this object).

Functionality provided for access rights management includes creating, updating and deleting roles, and adding restrictions to a role and removing restrictions from a role, as well as enforcement of these access rights at run-time.

<table>
<thead>
<tr>
<th>Microarray User</th>
<th>Experiment procedure</th>
<th>Microarray</th>
<th>W</th>
<th>Other</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Experiment context</td>
<td>Microarray</td>
<td></td>
<td>Other</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Software entity description</td>
<td>ANY</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computational processing</td>
<td>ANY</td>
<td>isDefinedBy.name = ‘$User.name’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Session information</td>
<td>ANY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>User information</td>
<td>ANY</td>
<td>W</td>
<td>name = ‘$User.name’</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Access rights</td>
<td>ANY</td>
<td></td>
<td>Other</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Information data types</td>
<td>Microarray</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td>ANY</td>
<td></td>
<td>Other</td>
<td>RW</td>
</tr>
</tbody>
</table>

Figure 4.25: Restrictions for the \textit{Microarray User} role
4.6 Methodology for Integrating New Domains and Applications in CEE

Lack of a methodology for supporting new scientific domains and applications in a CEE causes inadequate transfer of expertise and hence inefficient utilization of efforts. Availability of a well-defined methodology for integrating new scientific domains and applications in CEE was also mentioned among the requirements for the base VL infrastructure that were presented in Section 2.6.

In this chapter, several different aspects of a CEE were addressed with a focus on its information management platform. More specific, an experiment model is proposed for uniformly representing diverse scientific experiments. Furthermore, several data models for modelling different types of information related to scientific experiments, and functionality for the management of this information are provided. The results obtained and the experience gained by the work described in this chapter are used for defining a methodology for integrating new scientific domains and applications in CEE [165]. The methodology provides a step-by-step guidance to domain experts, tool developers, and administrators during the process of adding new domains, applications, resources to the CEE.

The methodology provided in this section is generic and reusable; it can be uniformly applied to different scientific domains and to different scientific applications. Formulation of the results obtained by the work presented in this chapter in terms of a generic and reusable methodology also proves the genericness and reusability of the achieved results, which were among the main objectives of this dissertation.

Following subsections describe the methodology in details. Application of the methodology to a real-life application, namely to the DNA microarray application, is demonstrated in Chapter 6.

4.6.1 Integration of a New Domain in CEE

Integration of a new domain in CEE covers the development of a domain-specific application database, integrating and registering this database into the CEE and the information management platform, and defining and registering the domain users. Steps to take when integrating a new domain in CEE are enumerated below.

S. 1 Study different types of experiments being performed in the domain, to the level of detail of single components involved in the experiments and the related information for each component. This step is realized by a Domain Expert in close collaboration with an Information Management Admin (IM Admin).

S. 2 Generate process data flows (PDFs) for each experiment type. Discuss the PDFs with the experts from the domain, make any necessary modifications, and eventually confirm the PDFs. This step is performed by a Domain Expert in close collaboration with an IM Admin.

S. 3 For each PDF, map the PDF elements to the base data types in the EEDM, and model each PDF element by sub-typing one of the base EEDM data types.
The result is the process data flows with elements that are defined as sub-types of base EEDM types. These PDFs form the base for the application database schema for the domain. This step is performed by an IM Admin together with a Domain Expert.

S. 4 Cross-study different experiment types defined in terms of extended EEDM constructs, and identify the elements that are common to different experiments such as organism information. Identify the relationships between components in different experiments and these common elements. Confirm the result with experts in the domain. The result is the core schema for the application database. This step is performed by a Domain Expert and an IM Admin.

S. 5 Map the core database schema to a data definition language (e.g. Object Definition Language). Add the other types required by the information management platform: procedure data model, origin-copies, and context-procedure links. This step is performed by an IM Admin.

S. 6 Create the actual application database by loading the schema defined in the data definition language. This step is performed by an IM Admin.

S. 7 Register the database to the information management platform by providing the required information about the new data source. This step is performed by an IM Admin.

S. 8 Define and register the domain users using the user management functionality. This step is performed by a CEE Admin.

4.6.2 Integration of a New Application in CEE

Once the new domain is integrated, new applications from that domain can be integrated. Steps to follow when integrating new applications are given below.

S. 1 Using the access rights management functionalities, define different user roles and restrictions for each role on the data types of this domain. Assign one role to each domain user. These activities are performed by a Domain Expert.

S. 2 Using the Procedure Editor, define experiment procedures for different types of experiments that will be offered to users of this application. This activity is performed by a Domain Expert.

S. 3 Develop software entities for application specific functionality, such as an analysis tool for microarray data. This activity is performed by a Tool Developer.

S. 4 Register the software entities to the CEE. After the quality control applied to software entities, store the software entity definitions in the information management platform using functionality for computational processing management, and store the executables in the CEE repository for software entities. These activities are performed by a Domain Expert.
4.7 Conclusions

Experiment-related information in a CEE includes information about scientific experiments (e.g. experiment procedures, experiment contexts, computational processing of experiments), information about users (e.g. user information and access rights definitions), and CEE-related information (e.g. information about active sessions, information about functionality provided by the CEE). The main subject of this chapter was to define a framework for managing information related to scientific experiments in a CEE. In addition to the data modelling for experiment-related information and functionality modelling for the management of this information, the framework also covered an experiment model and framework for user environment. In specific, this chapter:

1. defined an experiment model that is capable of uniformly representing different aspects of heterogeneous scientific experiments,

2. described a user environment for uniformly presenting the experiment-related information to users and functionality for managing this information, where the uniformity allows for the reuse of facilities and functionality for supporting different types of experiments,

3. defined several data models for representing the experiment-related information, which are uniform and reusable (to model information related to heterogeneous experiments) and open, flexible and extendible (to improve the developed models in the future when needed and to model new types of information),

4. defined functionality models for managing the experiment-related information in a CEE, which are generic and reusable (to support uniform management of heterogenous experimental information), and

5. defined a methodology for integration of new domains and applications in CEE based on the results obtained and experience gained by the work presented here, which provides a step-by-step guidance to domain experts, tool developers, administrators during the process of adding new applications to the CEE.

The data and functionality models defined as part of the framework described in this chapter are generic and can be implemented by different collaborative experimentation environments in different ways. Chapter 5 presents VIMCO, the information management platform of the VLAM-G virtual laboratory environment, as one specific implementation of the information management framework described here.