COPPE

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

GERÊNCIA DE RECURSOS CIENTÍFICOS: APOIANDO A REALIZAÇÃO DE EXPERIMENTOS *IN SILICO*

(SCIENTIFIC RESOURCES MANAGEMENT: TOWARDS AN *IN SILICO* LABORATORY)

RELATÓRIO TÉCNICO ES-605/03

Autores:

Maria Cláudia Reis Cavalcanti
Marta Lima de Queirós Mattoso
Maria Luiza Machado Campos

RIO DE JANEIRO, RJ - BRASIL

JUNHO 2003

COPPE

FEDERAL UNIVERSITY OF RIO DE JANEIRO

SCIENTIFIC RESOURCES MANAGEMENT: TOWARDS AN *IN SILICO* LABORATORY

TECHNICAL REPORT ES-605/03

This work is an extended version of the thesis presented to COPPE/UFRJ as a partial fulfilment of the requirements for the degree of

Doctor of Science (D.Sc.)

In Computer Science and Systems Engineering

By

Maria Cláudia Reis Cavalcanti

RIO DE JANEIRO, RJ - BRAZIL

JUNE 2003

SCIENTIFIC RESOURCES MANAGEMENT:
TOWARDS AN *IN SILICO* LABORATORY

Maria Cláudia Reis Cavalcanti

Advisors: Prof. Marta Lima de Queirós Mattoso
Prof. Maria Luiza Machado Campos

Department: Systems Engineering and Computer Science

One of the main challenges of scientific applications is to allow scientists to share their scientific resources. Besides scientific data, scientific programs and models are also valuable resources to exchange. Usually, scientific programs are available for local installation, in compressed files with configuration scripts. Data input and output files are rarely organized and in silico essays are frequently lost. The database community has been working on data management for the last four decades. However, managing models, programs and workflows as well as data is a new issue and complex task. To facilitate the exchange, reuse and dissemination of information we propose a Web services based architecture for managing distributed scientific resources. The main contribution of this thesis is an architecture (SRMW) and an enhanced metadata (SPMW) support system for effective management of distributed scientific resources. The architecture and metamodel have been studied under two real scientific applications evidencing important and innovative decisions on the design of SRMW and SPMW, such as: (i) definition of three basic metamodel categories, i.e., model, program and data; (ii) characterization of experiments and essays; (iii) publication and navigation of scientific resources using Web services platform.

Index

Figures

1. Introduction

Scientific experiments have traditionally evolved in isolation, i.e., scientists from different disciplines used to work on their own experiments. However, as science complexity increased over time, scientific experiments now depend on the cooperation among scientists from different disciplines and organizations.

Typically, scientists work with experiments based on scientific models, which are simplified representations of real phenomena. A scientific experiment can be viewed as a flow of data transformations that starts from raw data and finally produces data with added scientific value. Therefore, programs and data are some of the most valuable scientific resources at scientific laboratories. These labs usually have multiple versions of a single program, as well as multiple formats of a data set (images, flat files, databases, etc.). Also, multiple data sets may be used as input to those programs. Moreover, due to technology improvements, scientific data from different sources became largely available in digital media. Scientists can take advantage of such data availability by using them to enhance their experiments.

All these versions, formats and experiments are not easy to manage. As the number of programs grows, scientists find it more and more difficult to manage such resources. One of the reasons is the complexity of new programs coming up in the scientific scenario. Another difficulty is the natural need for composing chains of programs by combining their output/input data. In addition, the use of script languages is very popular within scientific communities. However, it does not facilitate the cooperation among scientists. Since scripts are not remotely accessed and usually demand programs to be installed locally, scientists cannot share their experiments with other scientists.

An experiment may begin when a scientist selects models and relevant input data for the problem to be studied, determining or developing an adequate flow of programs that can process the selected input data. Many of these programs are implemented for some specific platform, such as high performance and parallel machines. In this scenario, the main difficulties begin with trying to find the right program for each experiment and the interoperability with other scientific resources, such as programs and data. To find the right program means first to find the right model and this may not be a simple task. Information about the applicability of a model can feedback its users with more accurate model pre-conditions. Furthermore, to fully understand a model, the scientist may need to investigate previous case studies that have successfully used that model.

Usually, it is the scientist's previous experience that guides the choice of a model for a new experiment. To take advantage of such knowledge, the scientist should be able to access documentation on previous experiments. This documentation is not always available and may not be described within a common framework. Specially when dealing with empirical models, the scientist has to analyze contextual details of such experiments, verifying similarities with the problem in hand.

In summary, collaboration among scientists is based on the exchange of not only data but also scientific models, their implementations (programs), program compositions and experiment results. Therefore, scientists need an environment that supports geographically distributed team collaboration, and that enables scientific resources exchange between different teams. Ideally, a distributed information management architecture should enable scientists to publish (that is, make publicly available) their scientific data, models and programs. Program providers should be able to make their programs available to other scientists, describing and guiding their remote use. In addition, data providers should be able to make their data sets available to other scientists by describing data structure and providing the necessary means to get them. On the other hand, scientists and decision-makers should be able to search, select and manipulate published data, models and programs that are relevant to their experiments and decisions.

In order for distributed scientific resources to become part of a large information system on the Internet, they must be located, understood and efficiently accessed over the network. Sharing scientific data requires identifying not only data but also what model and model implementations (programs) are useful, where these programs are located and when (in which order) should programs be executed. It also requires enabling remote data access for execution at program locations. Moreover, based on this distributed approach, scientists should be able to configure their own combinations of programs provided by different teams.

A step towards this direction includes providing program and model descriptions to facilitate the selection of the appropriate model and consequently the appropriate program. However, this is not a simple task. First of all, model developers come from many different areas, dealing with different kinds of models and description standards. Also, each model description should include its use conditions, i.e., contextual and operational constraints, which are difficult to formalize. Now, suppose a user understands a model and selects it, finds the correspondent program and runs it. Feedback information on such model usage is valuable, as other users may need to investigate previous case studies involving that model to fully understand it.

In the past, several technologies have been proposed to address these issues. Among the more important ones are Heterogeneous and Distributed Database Systems (HDDS) (SHETH, A. P.; LARSON, J. A., 1990), Model Management Systems (MMS) (GUARISO, G.; HITZ, M.; WERTHNER, H., 1996) (BANERJEE, S; BASU, A., 1993) (BENZ, J; HOCH, R., 1999) (BRAZ, M. H., MELO, R. N., 1989) and Scientific Workflow Management Systems (WESKE, M.; VOSSEN, G.; MEDEIROS, C., 1996) (AILAMAKI, A.; IOANNIDIS, Y.; LIVNY, M., 1998). These approaches are not sufficient to address all those issues simultaneously, however, their combination can be helpful.

We present the combination of the desired functionality through an architecture and a metamodel using Web services (WS) technology. Data and programs can be published as Web services. Web services classes may be used to categorize and classify data and programs. These classes can be used in service composition, which can become a workflow. However, this is not a simple task. Aggregating equivalent programs and data means to overcome their heterogeneity. The Web services Description Language (WSDL) (WSDL, 2003) provides an abstract level for program and data resources descriptions. However, WSDL is not sufficient to address the heterogeneity problem, specially in the scientific environment, where programs are based on scientific models. Since WSDL was originally proposed for generic service description, it lacks application-related semantic descriptors. To aggregate scientific resources we had to extend WSDL to provide a better metadata support.

The main goal of our approach is to provide a Web services environment, to deliver semantic information about these scientific resources. We propose the Scientific Resources Management (SRM) architecture where scientific users can remotely access and share programs and data, as well as scientific workflow definitions and experiments. To make these resources really useful, SRM embeds the Scientific Publishing Metamodel (SPM) to harmoniously describe them. SPM relates models, programs and data through specific categories and semantic relationships. SRM is then implemented as an enhanced Web services architecture (SRMW) where SPM is represented as an extension of WSDL (SPMW). SRMW architecture provides seamless interoperability among published data and programs. In addition, equivalent programs can be described and grouped in one service.

The main contribution of our work is to combine a metamodel-based architecture for managing scientific resources to a promissing and adequate technology, i.e., Web services. Through this combination, we successfully address what HDDS, MMS and WfMS approaches individually fail to address. In addition, our approach is an innovative contribution to the scientific area.

We have exercised SPMW metamodel, applying it to scientific resources of two different research teams. The first one took place at the Petrobras Research Centre (CENPES), with the collaboration of the biocorrosion team of specialists. The second one took place at the Institute of Biophysics Carlos Chagas Filho (IBCCF), UFRJ, with the collaboration of dynamic molecular biology specialists. In both teams, scientists and specialists can browse metadata that describe scientific resources to find useful information for their research projects. Then, they can use these resources published as Web services to perform new experiments.

We have shown that our metamodel-based approach has provided a crucial support to scientific applications development. In our approach, several conceptual levels are captured in SPMW metamodel specially designed to those applications, and scientific resources are described according to this metamodel. One of the main contributions is the distinction between models and their implementations. Also, like in traditional scientific laboratories, SPMW is able to register the ongoing experiments.

This work is organized as follows. In section 2, we have briefly described some scientific applications. These applications have motivated our work on providing a scientific resources management solution. Then we discuss the main scientific resources, characterizing each one of them. Finally, we close the section with the identification of requirements for an adequate solution.

In section 3, we have described some of the current main approaches that address scientific resources management. Some approaches include recent technologies, while others embed these technologies in relevant research projects. We organize them according to their focus, in three subsections: handling the distribution and heterogeneity of scientific resources; describing scientific resources; managing scientific workflows and registering their usage. Finally, we close the section with an analysis of the solutions presented so far.

In section 4, we describe our solution to scientific resources management. The functionality of the SRM architecture is described in modules. Each of SRM modules is described in detail. As SRM is a metamodel based architecture, its metamodel (SPM) is also presented in this section. SPM is represented as a UML diagram, which is explained in parts. Subdiagrams are extracted from the main diagram, and each of the concepts is defined and exemplified. Finally, we analyse SRM in the light of related work.

In section 5, we introduce SRM as a Web services based architecture. We revisit SRM modules explaining how we have implemented them. To cope with the implemented architecture, SPM is expressed as an XML Schema (SPMW), which is also presented in this section. At the end, we explain how SPMW extends WSDL documents, by referring to its elements.

In section 6, we have described how SRMW can be used. We have instantiated SPMW documents according to real scientific application resources. First we describe resources of a biocorrosion application and then, we describe resources of a structural genomic application.

Finally, we conclude this work summarizing its contents and including some perspectives for future directions. Also, we have included an appendix with the complete SPMW XML schema.

2. Scientific Resources Management

A scientific application can be defined as a computer application that addresses a specific science investigation, typically astronomy, biology, physics, engineering, geology among others. The main users of such applications are scientists, some playing the role of application developers, and others playing the role of final users. Usually, these applications are computer programs or systems developed within some research laboratory tools, such as pollution control systems, molecular dynamics simulators, astronomy image processing programs, weather forecast systems, etc.

Throughout the years, scientific applications evolved together with computer technology. Nowadays, scientific applications involve different resources, such as complex programs and large data sets. However, scientific groups have difficulties to organize their application resources. Some of these applications use many computer programs that may have different versions, which may derive from different abstractions, or models. It is also a hard task to manage data sets used as input and produced by those programs. In addition, combining sequences of programs is also a usual requirement. Therefore, the increasing complexity of scientific applications and the difficulty of dealing with the diversity and quantity of the scientific application resources have driven scientists to develop scientific resources management solutions.

The objective of this section is to identify the requirements for scientific resources management through the analyses of typical scientific applications and the identification of the necessary resources. We have chosen two typical scientific applications examples: environmental and biophysical. Both applications make use of different scientific resources, which are identified and characterized.

Sections 2.1 and 2.2 briefly describe environmental and biophysical applications, respectively, and include some illustrative examples, where some of the difficulties faced by these applications are identified. Particularly, we were able to examine closely a biocorrosion application, which was part of a research project called SIMBIO (MOURA, F.A., 2001), as well as a structural genomic application, which is now part of an ongoing research project called MHOLline (RÖSSLE, S., RIBEIRO, S., *et al.*, 2002). Section 2.3 identifies which resources are typically handled by scientists, and defines them.

Finally, section 2.4 identifies the requirements for providing a solution for scientific resources management.

2.1 Environmental Applications

Environmental applications are designed to represent environmental systems. Environmental systems involve elements of the Earth surface and their relationship, which means that spatiality, is an inherent characteristic. These systems focus on the structure, functionality and/or dynamics of the elements they involve. Considering the variety of elements involved and also the different ways they may relate to each other, it is easy to envision the complexity of such systems.

Environmental systems may be divided into two broad categories: Ecosystems and Geosystems (CHRISTOFOLETTI, A., 1999). The former one concentrates on ecological issues related to biologic communities, their habitat and characteristics, while the latter concentrates on the geographic distribution of the elements. Ecosystems are defined as systems that cover a relatively homogeneous area of live organisms interacting with their environment (non-living elements). The living beings are the main elements of such systems. Among the relationships between the elements present in these systems are the energy exchange, the nutrients exchange, the productivity, the population dynamics, etc. For geosystems, also known as physical environmental systems, the geographic aspect is the most important. Whichever elements are present in these systems (climate, topography, rocks, water, vegetation, animals, soils, etc.); they are always related to some point in space.

The understanding and solutions for environmental issues involve the design of applications that use many other techniques, such as numerical analysis, computing optimisations, econometric evaluations, etc. For instance, in order to solve some problems related to pollution, like oil spills or gas leaks, it is necessary to use wind or ocean stream numerical calculations to predict the oil/gas behaviour and calculate the affected areas. The term *environmental* applies whenever these techniques are used together with a major objective: the evaluation of human impacts. For example, the ocean stream analysis is a specific theme of oceanography. However, when applied to the study of oil spills and its effects over the ocean life, it gets an environmental perspective.

The inherent complexity of environmental systems is due to the many elements and processes involved, and it can be addressed by specific disciplines, such as, geomorphology, climatology, geology, biology, meteorology, physics, chemistry, etc. Therefore, it is difficult to find a single environmental specialist, because a person rarely gets skilled in that many disciplines. Usually, what happens is a natural separation of specialists, each one working on a slice of the same environmental problem. For instance, biologists work on biocorrosion of oil pipes and oceanographers work on ocean stream behaviour, but both may be involved on the same environmental problem: an oil spill from underwater pipes. They may be working at different agencies of the same company, or even in different companies, focusing on different aspects of the same problem. Therefore, environmental applications are usually a combination of programs developed by different specialists. Some typical environmental applications are described in the next sub-sections.

2.1.1 Farm Field Damage Prevention

Ailamaki et al. (AILAMAKI, A.; IOANNIDIS, Y.; LIVNY, M., 1998) present a typical example of the combination of programs and data from different disciplines. Atmosphere and soil specialists cooperate through the use of meteorological programs and data to build a scientific application to be used in the prevention of overnight frost damages in cranberry bogs, as Figure 1 shows. This application is used regularly to monitor temperatures. Initially, the Atmospheric Sciences Department uses an US forecast program to provide a twenty-four hour forecast of the atmosphere temperature based on satellite and ground information. Then, a second program (Bog Forecast extraction) uses this output to generate the temperature forecast for twenty-five meters above the vine locations. This output is then sent to the Soils Sciences Department that processes it using the CranEB program to derive a forecast for the canopy level. Later in the day, as new weather observations become available, the initial twenty-five meters forecast can be updated by a statistical analysis program, which compares CranEB output forecasts with new observed weather conditions, and provides corrections to the original twenty-five meters forecast. Then, CranEB is rerun to produce updated canopy-level forecast, which is fed into a visualization tool, which generates forecast graphs.

In this application, scientists were able to cooperate with each in order to understand each others programs and combine them harmoniously. However, program combination is a frequent issue in environmental applications and program combination must be addressed more effectively.

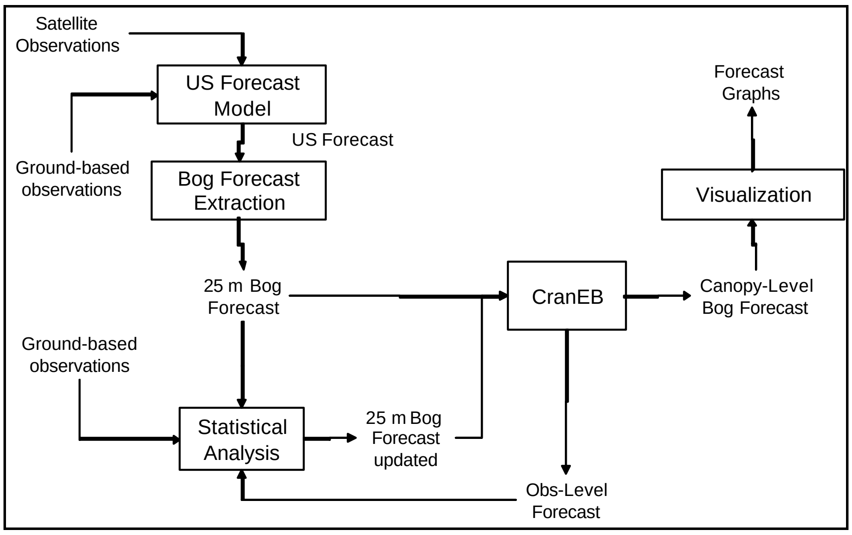


Figure 1: The Cranberry Application (AILAMAKI, A.; IOANNIDIS, Y.; LIVNY, M., 1998)

2.1.2 Diagnosis and Prevention of Biocorrosion on Petroleum Production

A research group working at the Petrobras Research Centre (CENPES), in Rio de Janeiro, is dedicated to monitor of biophenomena in the corrosion process of oil pipes. These scientists' activities are organized in case studies. Each case study involves the investigation of the affected region, some laboratory and computational analyses. Either the observation of a possible sign of biocorrosion, a prevention study or even a simple investigation may start a new case study. First, scientists collect water, soil or pipe samples from the region under investigation. There are regions where sampling data is provided by special field sensors. Alternatively, laboratory analyses provide numerical data sets from manually collected samples, such as chemical components' indexes. These data sets are then interpreted or analysed by means of scientific formulas in order to derive new data, or some useful conclusion, such as "there is evidence of a certain type of bacteria", "there is no evidence of a certain type of bacteria, some other type should be checked" or "re-sampling is needed".

Biocorrosion scientists (e.g. biologists or chemical engineers) are usually guided by the investigation of previous archived case studies, where they keep documented the use of formulas, programs and data. In addition, data collected from field sensors eventually need some computational treatment. Although distributed sensors may continuously obtain data, this is not always true for all case studies. Raw data should be invariably treated by data cleaning programs, which are derived for example from interpolation techniques. The combination of resources such as biocorrosion analytical programs and statistical programs typically characterize biocorrosion applications, as Figure 2 shows.

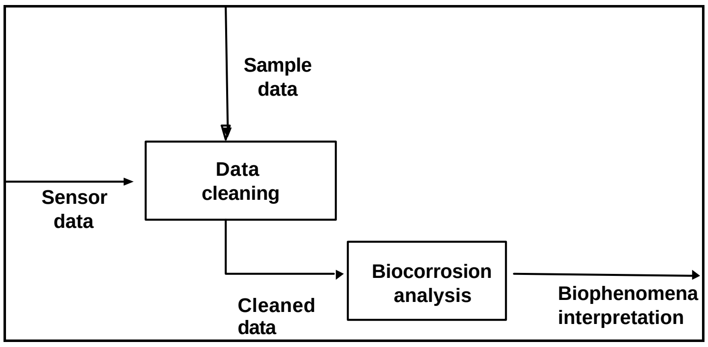


Figure 2: Biocorrosion Application

In biocorrosion case studies, to find the right formula for each case is the main difficulty, and program understanding is secondary. Therefore, for this kind of application, it is important to have a detailed description of models and access to previous case studies.

2.1.3 Air Pollution Forecast

At the National Institute of Research in Informatics and Automation (INRIA), in Paris, researchers had worked on a project called DECAIR (PROJECT DECAIR, 1999-2002). The aim of this project was to provide companies in charge of forecasting urban air pollution with good quality data derived from Earth Observation (EO) devices, in order to improve the results of existing air quality forecast results. The idea of the project was to provide support for the definition of scientific applications, combining satellite images treatment programs and air quality forecast programs. The development of such applications requires the collaboration of two kinds of scientists: those specialized in air quality forecasting and those specialized in satellite image analysis. In a typical air pollution control application, satellite data are delivered to a pre-processing program for image treatment, feeding two regional air quality programs, running in Berlin and Madrid, with EO-derived input data, as Figure 3 shows.

In this kind of application, the combination of programs is also an issue. It is required to understand programs and their constraints, such as which geographic area they were built for.

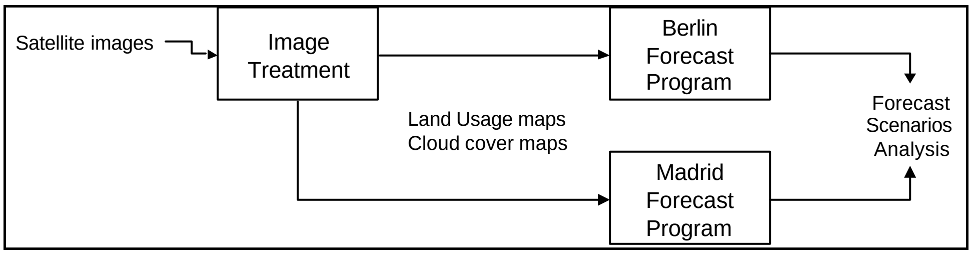


Figure 3: Pollution Control Application

2.2 Biophysical Applications

In the last decades, biochemical laboratories have been developing scientific applications. Programs and data are some of the most valuable scientific resources in these labs. They usually run exhaustively a single program with different data sets. This single program can be available at different sites, in different versions or formats. The data sets can also vary in versions, as well as multiple formats (images, flat files, databases, etc.). Typically, each program is studied and installed by one or two scientists. Other scientists, who are not familiar with these programs, usually request specialists to configure and run these programs, using their skill. As these program executions increase, the number of data sets used as input and produced by those programs also increases. Besides, each of these input data sets has a particular format and has been prepared by one specific scientist.

As the number of programs increases, scientists find it more and more difficult to manage such resources. All these programs, data versions and formats are not easy to manage. Even under rigid lab rules it is very difficult to keep track of these resources. Usually, scientists count on the file directory structure to organize data inputs and outputs, labelling them according to the research project, program or scientist. However, this was proved not to be sufficient as many of these labs have been hiring or consulting database specialists to provide solutions for their resources management problems.

More importantly, there is a natural need for composing chains of programs through the combination of their output/input data. Multiple possible combinations make biophysical applications even more complex. Structural genomic applications are a typical example of bioinformatics program composition that is described in the next sub-section.

The Institute of Biophysics Carlos Chagas Filho (IBCCF) of the Federal University of Rio de Janeiro is one of the few research centres that are developing structural genomic projects. These projects are producing a vast amount of protein sequences as data resources, emerging the need for using high throughput methods to predict structures and assign functions to these proteins. However, the analysis of several genome sequences indicates that the function of proteins cannot be inferred from a significant fraction of the gene products. In fact, isolate sequence homology searches do not always provide all the answers, since some proteins may not keep sequence homology throughout evolution. On the contrary, the molecular (biochemical and biophysical) function of a protein is tightly coupled to its three-dimensional structure.

A good approach that contributes to the prediction of three-dimensional protein structures is comparative modelling, which predicts a reliable structure for a sequence using related protein structures as templates. This approach consists of the following steps: finding known structures related to the sequence to be modelled; selecting related sequences as templates; aligning the sequence with the templates; building a model, and finally, validating the protein structure, as illustrated in Figure 4. There are several programs addressing each of these steps. To enable large-scale modelling the IBCCF is developing an application called MHOLline (RÖSSLE, S., RIBEIRO, S., *et al.*, 2002), which assembles these steps in an automated program sequence, using a set of strategically chosen programs.

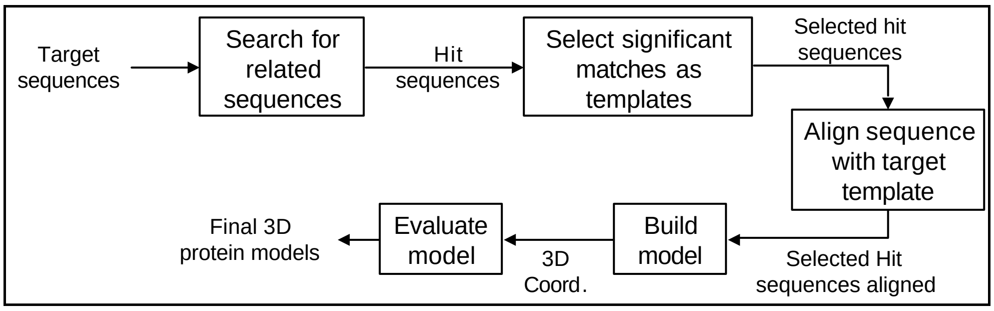

Target sequences
Search for related sequences
Hit sequences
Select significant matches as templates
Selected hit sequences
Align sequence with target template
Selected Hit sequences aligned
Build model
3D Coord.
Evaluate model
Final 3D protein models

Figure 4: Structural Genomic Application

In this kind of application, again it is clear the need for understanding programs, their constraints, and their background, such as the algorithm that originated it. Without this knowledge it is hard to provide support for program combination, as it seems to be the case of most scientific applications.

2.3 Defining Scientific Resources

The resources used in the scientific applications described in sections 2.1 and 2.2 are mainly programs and data. It is worth noting that these applications involve chained programs, which means several programs are executed in an organized way. These program chains are called scientific workflows. Once defined, these workflows can be reused by other applications, what turns them into valuable scientific resources.

It is important to note also, that most of the programs used in these applications, due to their scientific foundation, were developed based on a previous conceptualization. In the examples described before, some programs were implemented based on model representations such as: a formula, an interpolation technique, a forecast equation, an algorithm, etc. Within the scientific community, these are scientific model representations. Scientific models are even more valuable than the programs that implement them, and should also be considered as scientific resources.

Another important characteristic of the scientific community is the fact that their work is usually based on previous works and experiences. Daily, scientists deal with scientific models and programs, building or capturing input data sets and producing new ones, comparing results, tuning programs, repeating program executions, etc. All this activity requires documentation. In organized labs, scientists prepare written reports on a daily basis, registering models, programs and data used. These reports, that we refer here as scientific experiments, are also considered valuable scientific resources.

According to the scientific scenario described here we have identified five main scientific resources, which are: models, programs, data sets, workflows and experiments. The scientific applications described earlier are examples where these resources are exchanged among scientists within a research project. However, it may sometimes happen across projects. In this work, we are particularly interested in providing support for cooperation intra and inter-projects. To make these resources really useful for the scientific community it is necessary to fully describe them. Next sub-sections characterize each of these resources, based on the scientific community literature. Beginning with models, programs and data, we try to identify characteristics that would be useful to describe them.

2.3.1 Scientific Models

According to (HAGGET, P.; CHORLEY, R. J., 1967. as quoted by CHRISTOFOLETTI, A., 1999) a *model* is defined as a simplified abstraction of the reality and presents, in a generic way, characteristics or important relations between elements of such reality. Although models are highly subjective approximations of reality, since they do not include all the associated observations and measurements, they are considered valuable because, by hiding some details, they represent fundamental aspects of reality.

The term "model" is a constant source of confusion as it is used in many different contexts. Therefore, as our focus is on the scientist perspective, it is helpful to clearly define what we mean by "scientific model". Scientists are dedicated to the development of an understanding of how the natural world works, which is achieved through the conceptualization of models of various natural processes. Thus, a scientific model can be defined as a set of ideas that describe a natural process (CARTIER, J., RUDOLPH, J., STEWART, J., 2001). It is also important to stress that a 'scientific model' is distinct from its representation. Physical replicas of systems (e.g. solar system), formulas, equations, algorithms (e.g. image processing algorithms), graphics, and maps are examples of scientific model representations. In this work we are especially interested in those scientific model representations that might be implemented as programs and might be executed throughout the Internet. From now on, we will use the term model meaning scientific models that can be implemented as programs.

To provide for scientific resources exchange means that each resource needs to be described and this is also true for scientific models. The process of building a model, also known as modelling process, is a good starting point. Through the modelling process it is possible to identify the elements that best describe a model. Some guides for building models can be found in the literature (RICHARDSON, G. P. AND PUGH III, A. L., 1999) (GOLDBARG, M., LUNA, H., 2000) (SPRIET, J. A., VANSTEENKISTE, G. C., 1982) (HAEFNER, J. W., 1996) (GRANT, W. E., 1986), where the modelling process includes three main steps: problem identification, formulation and validation. Problem identification includes the awareness of a problem and an unambiguous definition of it, by describing the context, listing symptoms and stating model purpose. Formulation means to delimitate the system to be modelled, by identifying the elements to be focused, establishing the system boundary, raising hypotheses and representing them. The validation step involves the implementation of the hypothesis (e.g., in some programming environment), testing and validating it for correctness. The validation step is used to check model results with real data. If any inconsistency is found in this step, one should go back to the second step, reformulate hypotheses, etc. In the last decade, some techniques have been used to implement models, such as programs, fractals, expert systems, fuzzy logic, neural networks, etc. Feuvrier (FEUVRIER, C. V., 1971) states that a model substitutes the reality it simplifies. It is important to have in mind though, that no matter how precise it is the technique used to build a model, it is never a substitute to reality.

Some of the main elements for describing scientific models identified are: input and output variables, application area, scope, purpose, constraints, precision, hypothesis, parameters, classification, and bibliography. Some models can be derived or calibrated based on other models. The information about the relationship between models is also important to describe. Therefore, model derivation and calibration are also elements to consider in model description. All these elements are described in more detail along this section.

While formulating a model the scientist defines which variables are relevant to the problem at hand. These variables take part on the structure of the model, both as input and output. When formulating a model, the scientist first deals with the model conceptualization. At this point, model input and output are not committed to any form of implementation yet. Instead, model data input and output can be described in terms of the quantities to be considered, such as length, time, mass, temperature, pressure, energy, moment of inertia, force, etc. For instance, the temperature quantifies (measures) the intensity of the heat, i.e. the hotness or coldness of an object, such as a water sample. After figuring out which quantities to consider, the scientist starts implementing and testing the model, by taking sample objects and measuring them. The numbers used to express the model quantities are magnitudes, which are usually expressed as a multiple of a standard unit. For instance, a magnitude may represent the temperature expressed in degrees Celsius.

A variable of a scientific model may be not quantifiable. In this case, it may be described in terms of its classification. The values assumed by the corresponding implemented variable are expressed according to the formats that are used to express such class of objects. For instance, an image treatment algorithm may deal with the class of raster images. A raster image can be defined as an abstraction of a real world image where spatial data is expressed as a matrix of cells or pixels, with spatial position implicit in the ordering of the pixels. The implemented variable that corresponds to the model variable may assume values on a specific raster format, which may be, for instance, the bitmap format (.bmp).

A scientific model is usually associated to an application area. Some examples of model application areas are: industrial, economic, social, political, environmental, etc. More specifically, a model is conceived to address a specific target within its application area. The target or scope of a model corresponds to the natural world system/process it represents. For instance, a model scope may be a specific hydrographic basin, a geographic region, or an enterprise.

Each model has a specific purpose, for which it is designed. In general, every model purpose is to understand some slice of reality. In their work, Richardson et al. (RICHARDSON, G. P. AND PUGH III, A. L., 1999) state that the model formulation is guided by its purpose and present many examples of purpose statements. A statement of a specific production/distribution model says "... examination of possible fluctuating or unstable behaviour arising from the principal organizational relationships and management policies at the factory, distributor, and retailer. ...".

A model is not exactly true or false, its value is judged according to the contribution it brings to explain the system it represents (FEUVRIER, C. V., 1971). Precision is the measure that expresses how much a model is faithful to the system it represents. The greater is the model fidelity level the better it represents the system, and higher is its precision. For simplification purposes, a model is built considering some reduction factors. These factors are also known as model constraints. For example a "transportation model" may not consider traffic jam or a possible mechanic problem. Another example of model constraint can be found in a simple model of the moon's orbit around Earth, where geology details of both space bodies are not considered, but just their masses and the distance between them (GLEISER, M., 2002). Although a model is built based on restrictions, the quality and volume of the experimental data used in its construction may determine its precision. In summary, a model that has a solid empirical base guarantees better precision (FEUVRIER, C. V., 1971).

Every model is initially a hypothesis. Building a model represents the expression of a scientific hypothesis that needs to be validated (CHRISTOFOLETTI, A., 1999). There are two basic approaches for model validation used in scientific research: the deductive and the inductive. The deductive approach usually starts with a model, assuming some hypothesis, while in the inductive approach it is assumed that any pre-conceived idea or hypothesis would ruin the necessary impartiality of a scientific investigation. The inductive approach involves the study of collected data and derivation of many generalizations. Then, if any generalization is validated, it becomes a law. In the deductive approach, if the initially stated hypothesis is validated, it becomes a law. Otherwise, there might be a reformulation of the model or a new model is created.

Considering that systems evolve with time, models that represent them may get old. Therefore, models loose precision as they age. It is a usual procedure to update models or to adapt them to new targets. This procedure is known as model derivation. Scientists derive new models based on studies over some existing model. A study might consider, for instance, the influence of a new variable such as temperature.

Another important concept with respect to a scientific model is called calibration. To calibrate a model means to "tune" it precisely for a particular situation. Calibration is in a sense customizing a generic model. A generic model is conceived to address a generic system. However, it includes variables that assume values according to particular systems. These variables are referred to as parameters, to differentiate them from the other model input variables. Usually, a default value is provided. The calibration process usually involves multiple executions of the model with normal input for which the output is known, to provide the parameter estimation. The model output is then compared to real data and a correction factor is found. This value is then used as the parameter value. Sometimes these values are also called model constants as they become invariable for a particular situation. Frequently, the calibration of a model involves a lot of time and effort. Therefore, a calibrated model may be as useful as its generic version.

Traditionally, when selecting a model to use, scientists refer to its scientific publications and related explanatory material. In special, the scientific publication plays an important role on the model credibility. Therefore, a scientific model description should include or point to its bibliographic information.

Another helpful criterion for describing a model is its classification. There are many different proposals for classifying scientific models. Feuvrier (FEUVRIER, C. V., 1971) classifies them mainly according to logic and mathematic categories. Algorithmic, simulation, deductive, and probabilistic models are examples of model subcategories. Other aspects are also considered to provide more specific classifications, such as time dependence (dynamic and static models), stability (stable and unstable models), openness (open/closed models), durability (change-prone/long-lasting models), etc. The model classification presented by Banerjee and Basu (BANERJEE, S; BASU, A., 1993) groups models according to their goal, while other authors (ECOBAS) classify them according to their application domain area. Christofoletti (CHRISTOFOLETTI, A., 1999) presents various model classifications adopted by different domain areas: geomorphology, hydrology, climatology, etc. Possibly, there are many other classifications proposed. We consider it is important to use some classification standard to describe scientific models. However, due to the difficulty of achieving a consensus on model classification, we do not propose the use of a specific one.

2.3.2 Scientific Programs

The implementation of a model is part of the validation step of the modelling process. Scientific programs can be defined as scientific model computational implementations. As scientific programs are valuable and exchangeable scientific resources, they need to be described. A conventional program description is usually based on its interface, i.e., its input and output variables. A scientific program can also start to be described likewise. However, the scientific program description is associated to the correspondent model description. For each model variable, there is a corresponding program variable. The variables described in terms of quantities and classifications at the formulation step, are now program variables that are described in terms of units, type and format. This is also true for model parameters.

A programming language usually supports a variety of data types. Each program variable is associated to a data type that will assume numeric or non-numeric values. Numeric values are magnitudes used to represent model quantities, expressing them as multiples of a standard unit. For instance, a magnitude may represent the temperature expressed in degrees Celsius. In the case of non-numeric variables, the model variable is described through its classification, and the correspondent program variable assumes values on a specific format, which is processed by the program. Raster images, for instance, may assume different images such as windows bitmap image file (.bmp) and graphic image file (.gif). Therefore, a variable unit and format are also useful information to describe program variables. This information is usually hidden inside the program code as comments, and if not informed it may generate bad output data.

Although widely accepted for model implementation, computers may impose severe restrictions to the model performance and credibility. The well known problem on designing scientific programs is the natural difficulty of representing “real” numbers in a computer. No matter what programming language the scientist chooses, this problem will be present. Each programming language offers some limitation on the representation of real numbers, providing different levels of precision. They usually provide a data type that may store numbers with a limited storage capacity. In C programming language, for instance, it is possible to manipulate real numbers using one of two keywords to declare a variable: float and double. A float or floating point number takes four bytes to store and has about seven digits of precision, while a double, or double precision, number takes eight bytes to store and has about 15 digits of precision (MS C/C++). These representation limitations may result in a small imprecision. However, when a small error is repeated many times, it may incur in a big one.

To overcome such limitations, programmers usually make use of more computer memory and processing time. Therefore, scientific programs typically add parameters for controlling the use of computer resources, allowing the user to choose between precision and resource consumption.

The software engineering area has brought into the programming community several tools, such as maintenance and development tools, that typically support the programming activity. Especially in large development project teams, to manage program versions and their compilations demand a considerable effort. This is also true for the scientific community. Besides its interface, scientific programs may also be described by information such as author, programming language, version, compiled code, host operational system, hardware configuration, etc. For the scientific community, to learn about the computer resources where an executable code is hosted is very useful. Multiprocessing capabilities, for instance, may determine how to better tune a program precision parameter to get more accurate results.

More than one scientific program may be implemented based on the same scientific model. We may say that a model description works as scientific programs classification. As a consequence, program variables are classified by model variables. This mapping is especially useful when searching for similar programs, while allowing for a richer program description.

2.3.3 Scientific Data

Typically, scientific data are usually measured data. High technology mechanisms allow for the collection of a vast amount of scientific data. Satellites are constantly monitoring the earth and the stars, while sensors and detectors are spread all over the Earth continents and oceans, both collecting large quantities of raw data everyday. In addition, as computers become more powerful, and sophisticated data transformation models are implemented, more data are generated. Summed to the conventional data, all these data are becoming more and more unmanageable. Some data management issues have been recently discussed in the Workshop of Data Derivation and Provenance (WDDP) (BUNEMAN, P., FOSTER, I., 2002). One of the position papers at the workshop (LESK, M., 2002) defines the term "data curation" to be the collection and maintenance in a useful form of large amounts of scientific data, and the software needed to facilitate the usage of this data by others. Data curators should be able to provide information about data provenance, such as data version, generation mechanism and date. The generation mechanism may correspond to a code execution. Through this information it might be possible to trace back data derivation, identifying related data sets, which were used to derive those data, up to the raw data that started the transformation process.

Data derivation information, also known as data lineage, is not always sufficient for its complete understanding. As a transformation model is a simplification of reality, it is important to be aware of the reduction factors, when using it, and therefore, provide a careful interpretation of its results.

Besides data provenance, data structure is other important information for describing scientific data. In general, most database management systems provide structural information in the form of a data schema. This information is highly useful, however, to provide richer semantic information about data demands more than the data schema. Especially with respect to scientific data, there is the need for providing information about the units and formats used to represent those data. As we have seen before, scientific programs descriptions include variables specific units and formats. To turn data sets into exchangeable scientific resources within the scientific community, data descriptions should be similar to program variable descriptions. Thus, it becomes possible to compare scientific program and data descriptions, and verify if they match. In WDDP we also presented a position paper that addressed these metadata issues (CAVALCANTI, M., CAMPOS, M. L.; MATTOSO, M., 2002).

2.3.4 Scientific Workflows

According to the Workflow Management Coalition (WfMC), a workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (WfMC-a, 1999). The workflow definition consists of a network of activities and their relationships, some criteria to indicate the start and termination of the process, and some information about individual activities, such as participants, associated applications and data, etc (WfMC-a, 1999). A large number of Workflow Management Systems (WfMS) are commercially available, each one providing different workflow definition languages (VAN DER AALST, W. et al., 2000). These systems are able to interpret these definitions and execute the workflow, generating what is known as a workflow instance.

Typically, scientific applications rely mainly on program use expertise, where scientists combine different programs with the objective of solving application problems. Environmental and biophysical applications are examples where these combinations are frequent. Environmental applications are typically multidisciplinary demanding the combination of expertise from different disciplines, while in biophysical applications, biologists depend on the combined use of bioinformatics expertise.

In order to reuse them, scientific programs compositions can be documented as exchangeable scientific resources. A workflow definition language can be used to define these scientific workflows. Each scientific program corresponds to a step or task of a workflow specification, which defines a certain procedural logic (step precedence, loops, conditions and parallelism). In particular, the empirical nature of some experiments demands some sort of tight control. Often, certain steps are not successful and have to be re-executed, leading to unexpected loops in the process.

Scientific workflows are different from business workflows (LEYMANN, F., ALTERHUBER, W., 1994). Some of the main characteristics that make clear this difference have already been identified (MEDEIROS, C, VOSSEN, G.; WESKE, M.; 1995). First, scientific problems are usually complex and not known in advance, and hence tasks are frequently not predictable, which means that *ad hoc* workflows are a common practice. Indeed, workflows are typically built on demand by final users (i.e., scientists) in contrast to previously planned business workflows, which are built by specialists or system administrators.

Another important characteristic is that scientific workflows are highly changeable while business workflows are usually invariant. Since scientific processes in general are not fully specified before they start, scientific workflows should have flexible definitions, enabling a scientist to modify some of its steps. This change may simply consist of choosing an alternative program to perform a given step of the workflow. For example, various image analysis techniques can be used for a given image depending on its accuracy and the context in which the image was taken (e.g., meteorological conditions, and date). The choice of a given program usually depends on program constraints directly associated with its input (e.g., meteorological condition, cloud-cover maximum). These constraints can be verified for already existing data sets. However, this is not possible for those data sets that have to be computed on-demand, which are computed by the execution of previous steps of the workflow. Thus, the choice of program has to be done incrementally, backward or forward, along the execution of the workflow.

Finally, a third particularly significant characteristic of scientific workflows is the user need for workflow instances reuse, differently from users of business workflows. The scientific process is strongly based on experimental investigation, evidence accumulation and result assimilation. Therefore, successful and unsuccessful workflow instances are interesting scientific resources.

2.3.5 Scientific Experiments

Scientific results should be disseminated and reused. Typically, scientists keep track of all the performed experiments, even if they have failed. This is because scientists learn from their past experiences, even if they ended up erroneously.

In this work, our focus is on what is called *in silico* experiments. This expression has become particularly popular within the life sciences area. It is an analogy to the Latin expression *in vitro*, which means "in glass", and relates to experiments performed in an artificial environment outside the living body, such as a test tube or culture dish. *In silico* means in silicon, and relates to experiments performed with the help of computer chips.

Although we have been focusing on *in silico* experiments, we can refer to a traditional definition of experiment (RUDIO, F. V., 1978) that states *"The experiment is a situation, created in laboratory, which aims to observe, under control, the relationship between phenomena. The word control is used to indicate that there were efforts to eliminate, or at least reduce, as much as possible, the occasional mistakes during an observation. These efforts are materialized as procedures … rigorous techniques, which aim at the control of variables that will be observed… The experiment is used to verify hypotheses."*

Based on this experiment definition we can argue that an experiment is associated to a set of controlled actions. These actions are usually similar to each other, and their results are commonly compared to each other to verify or not the experiment hypothesis. Scientists have well evolved in performing *in silico* experiments, building scientific applications where experiments usually involve the execution of scientific workflows. These workflow executions are analogous to the controlled actions of traditional experiments. Thus, within the scope of this work, we can define a scientific experiment as an *in silico* experiment that is associated to a set of scientific workflows.

A scientific experiment should keep track of the whole *in silico* experiment. To do this, scientific experiments should be able to document all instances related to the associated scientific workflows. This registry should include all scientific models, programs and data resources used in each workflow instance.

Finally, we may conclude that scientific experiments are also valuable scientific resources to be exchanged. Besides its association to a set of scientific workflows, and their correspondent instances, a scientific experiment should be described by other important information, such as the hypothesis the scientist is aiming to prove, the research project the experiment belongs to, as well as the scientist report containing the experiment annotations and conclusions.

2.4 Requirements for Scientific Resources Management

Inferring models from observations and studying their properties is really what science is about. (LJUNG, L., 1987). There is a vast amount of scientific models in the literature of many domain areas. Nowadays, scientists make use of different multimedia digital scientific resources. Especially after the emergence of computers, scientific models have become easier to infer, which increased the scientific production considerably. As a consequence, scientific data have become easier to generate. With the advances of computer technology, the scientific community has become more and more dependent on the use of models and programs as tools for understanding real phenomena. This dependency has determined the rapid growth of the quantity and variety of the scientific resources available. For instance, it is now possible to collect large amounts of data every hour from satellite sources, and these data can be transformed by computer models into new data. Even within a specific domain area it has become difficult to manage the amount of scientific models available and all the related resources that come with them, such as programs, data, experiment reports, etc. In addition, research project teams are frequently international or inter-institution, making scientific resources management even harder.

Scientists need an environment with several facilities such as: to support geographically distributed team collaboration, and to enable scientific resources exchange between different teams. Program providers should then be able to make their programs available to other scientists, describing and guiding their remote usage. Moreover, based on this distributed approach, scientists should be able to configure their own scientific workflows by combining programs provided by different teams. On the other hand, data providers should also be able to make their data sets available to other scientists by describing their correspondent data structure and providing the necessary means to get them.

There are many computer systems developed to deal with scientific resources. Systems like Simile (MUETZELFELDT, R. AND TAYLOR, J., 2001), Stella (RICHMOND, B., 1994), and many others (GEOFFRION, A. M., 1987) were designed to address the modelling process. In this work we focus on management systems, where models are already developed and implemented, and need to be exchanged.

Scientific resources are quite heterogeneous with respect to their storage models, schemas, semantic meanings, and processing environments. First of all, data sources can be provided according to different storage models: relational or object-oriented databases; files; spreadsheets; Web sites; etc. Consequently, access to data is also diversified, ranging from standard languages like SQL or OQL to specific protocols and APIs. Second, syntax and semantic conflicts may arise, because similar data may be stored according to different schemas, and similar schemas may represent data with different meanings. Third, model-based programs are available in a variety of processing environments: different versions, different software and hardware platforms. Managing the use of these programs means to support their remote execution, using data from elsewhere and generating new data that should also be available for access. The management of such distributed and heterogeneous scenario may face some hard obstacles, such as to host programs that are not remotely accessible, to host the results of program executions, to adapt data into programs input format, to filter data according to program constraints, among others. Therefore, in a multidisciplinary and distributed scientific environment, scientists and specialists need a scientific management system able to provide interoperability between heterogeneous and distributed programs and data resources.

More than to provide support for accessing different platform data and program resources, scientists need to understand how to use them. Furthermore, semantic integration is also required. To provide support for semantic description of scientific heterogeneous resources, several domain specific description standards have been proposed. However, a universally accepted standard has not yet been proposed. Indeed, it is hard to identify a complete and sufficient set of descriptors that may encompass the range of heterogeneous resources handled by the scientific community. Moreover, besides this set of descriptors, each scientific subarea may have its own set of specific resource descriptors. Future initiatives for scientific resources description should provide extensibility mechanisms. This extensibility guarantees that any special aspect of resources from different areas could also be included in its description. Therefore, scientific management systems need a generic and extensible description mechanism to address scientific resources description.

Typically, scientific applications usually require combining multiple mathematical and algorithmic model-based programs. Specially when addressing environmental problems, required models are usually composed by linked sub-models (SCOTT, E. M., 1996), originally from different disciplines. Also, in bioinformatics laboratories, scientists need to combine various different scientific programs. Scientific program and model combinations were defined as scientific workflows. However, to address the scientific community, and provide for scientific workflow management, traditional business WfMS are not sufficient. Scientific workflows require special facilities, such as to deal with incomplete workflow definitions, to allow for on the fly workflow definitions, and to register workflow instances.

When performing scientific experiments, the choice of a scientific model is usually guided by the scientist's previous experiences. Based on these experiences, the scientist can compare the use of different models to solve similar problems, find equivalent programs that implement the same model, choose the best program to the problem at hand, and more importantly, use the adequate parameter values. However, previous experiences are hard to reuse. The experience from a successful (or not) program execution is not always registered, and if registered, it is usually available only in paper reports. Consequently, scientists have difficulties in taking advantage from a large number of previous experiences. Therefore, scientific experiment registry is a fundamental requirement to scientific resources management systems.

In summary, to address scientific resources management, we have outlined four main requirements that these systems should provide support: scientific resources heterogeneity and distribution, scientific resources description, scientific workflow management, and scientific experiments registry.

In the next section we present the available technology and initiatives addressing each of these four issues. Scientific resources heterogeneity and distribution in section 3.1, scientific resources description in section 3.2, and both scientific workflow management and scientific experiments registry in section 3.3. We also comment on the main research projects that relate to each of these issues. In section 3.4 we discuss why these projects fail while addressing scientific resources management issues as a whole.

3. Scientific Resources Management Systems

Historically, the database community has initially addressed the scientific community. The first initiatives emerged around the 80's, when Model Management Systems (MMS) were proposed. These systems were specially developed for dealing with local scientific resources by providing functions, such as model description, model selection and model design. Borrowing ideas from the DBMS, MMS architectures maintain a model base (DOLK, D., KONSYNSKI, B. R., 1984). Basically, these architectures offer three modules: model design, model manipulation and model control, which interact with the model base. Model design supports the modelling process, which means building models. Model manipulation includes standard functionality of storage (insert, update, delete, display, etc.). Model control involves issues of access authorization, security and privacy, integrity, etc. Then, because of the diversity of models and problems to solve, model selection activity has emerged as a new module in the architecture, and the model manipulation restricted its functionality to model description (BANERJEE, S; BASU, A., 1993). Also, the importance of keeping track of previous experiments in an experiment base (GUARISO, G.; HITZ, M.; WERTHNER, H., 1996) increased the importance of the model control module, including experiment registry as one of its functions.

MMS alone does not address properly the scientific community. Frameworks such as distributed architectures, metadata management and workflow management systems should be combined, embracing MMS functionality, to address this community. In addition, technologies such as Web services and Grid computing should be the foundation of these frameworks. Scientific resource management systems need to provide solutions to four main problems: (i) how to handle the distribution and heterogeneity of scientific resources; (ii) how to describe scientific resources; (iii) how to manage scientific workflows; and (iv) register their usage. Next sub-sections discuss these four problems. Some of these approaches include recent technologies, while others embed them in relevant research projects. These discussions include presenting the most relevant technological foundations and representative research projects on each of the four problems. Some projects, such as MyGrid and GriPhyN address three issues, while others concentrate on one or two. Thus we separate the projects along the four problems. Problems (i) and (ii) are discussed in section 3.1 and 3.2, while problems (iii) and (iv) are discussed in section 3.3. On the last section we discuss the existent approaches and relate them to our work.

3.1 Handling Distribution and Heterogeneity of Scientific Resources

There are several technologies that have been proposed in the area of databases, information systems, and cooperative information systems that can be useful to address heterogeneity and distribution problems. The concept of information mediation, initially presented by Wiederhold (WIEDERHOLD, G., 1992), is one of the most important contributions to HDDS. It consists of defining an intermediate layer between information sources and applications. This intermediate layer provides an integrated view of information for queries without having to physically integrate data sources. So far, several mediator-based HDDS have been successful: Disco (TOMASIC, A.; RACHID, L.; VALDURIEZ, P., 1998), Tsimmis (GARCIA-MOLINA, H.; PAPAKONSTANTINOU, Y.; QuASS, D. *Et al.*, 1997), Garlic (CAREY, M. J.; ET AL, 1995) and Himpar (PIRES, P., 1997).

However, most HDDS usually focus on heterogeneous but structured data and do not address other scientific resources, such as programs. To deal with scientific resources, HDDS should be able to address also heterogeneous and distributed programs. In 1998, the Asilomar Report on database research (BERNSTEIN, P. *et al.*, 1998), the database community has proposed to focus on the problem of handling programs, besides data. A few years later, Le Select (LESELECT) was the first HDDS initiative to address directly programs distribution and heterogeneity. In the next sub-sections we present some recent initiatives on this direction. Le Select is discussed in section 3.1.1, followed by Web services and Grid (sections 3.1.2 and 3.1.3 respectively), which are more recent available supporting technologies for handling distribution and heterogeneity of data and programs.

3.1.1 Le Select

Developed at INRIA, Le Select (LESELECT) acts as a mediator-based HDDS. However, differently from traditional HDDS, Le Select was specially developed to support environmental applications, offering unique features to share both data and programs, while maintaining the general principles of mediator/wrapper architectures. Le Select implements a framework that facilitates the publication of distributed and heterogeneous data and programs, and provides common facilities to query published data and to invoke published programs (XHUMARI, F. *et al.*, 2000). Users publish data in their original format and location. There is no need for transformation or replication of these data. Similarly, programs remain installed in their original configuration and computer platform. Therefore, scientists may run their experiments, by feeding these programs with remotely published data, and by using programs from multiple disciplines, which are served in sites over the Internet.

Figure 5 presents the Le Select architecture. The intermediate layer, between information sources and applications, integrates information from multiple data sources without having to physically integrate them. In Le Select, data from each data source are wrapped into a common relational model of data. This is done via a piece of code called a data wrapper, i.e., publishing information of a given type (e.g., HTML file, C program or database) requires creating a specific wrapper for it. Each data wrapper interfaces with a local mediator called Le Select server, to form a publishing site, which is accessible from applications. When an application needs to access data from multiple data sources, it can connect itself to a Le Select client, which provides a JDBC interface to access multiple publishing sites (Le Select Servers) in a single SQL query. Facilities offered by mediators and wrappers enable the sharing of data without forcing each application to redundantly encode data transformation and data processing parts.

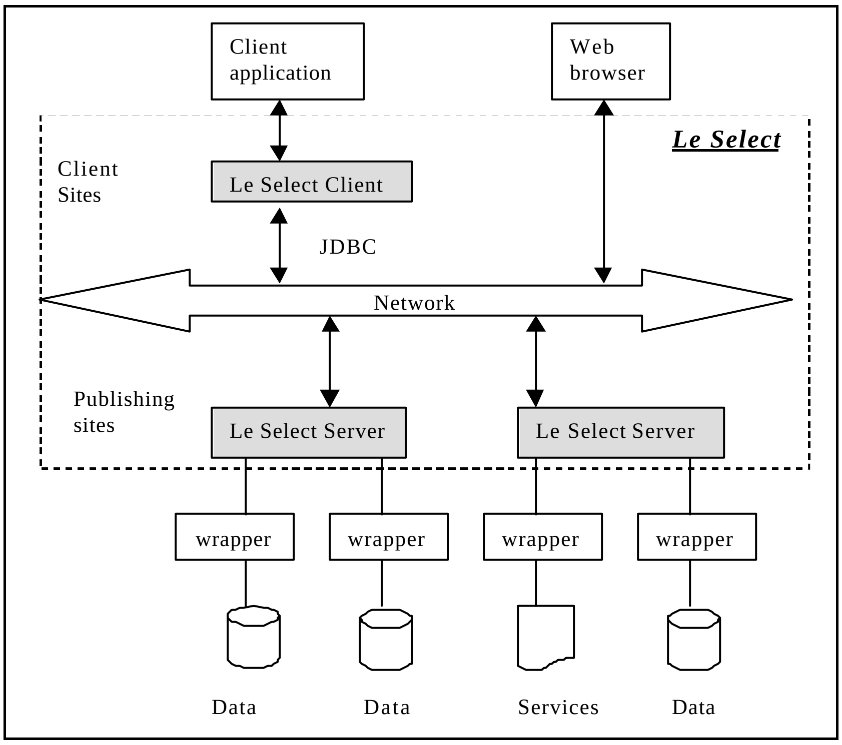


Figure 5: Le Select Architecture (XHUMARI, F. *et al.*, 2000)

Le Select also enables sharing services, which are available in a specific source, via a particular kind of wrapper, which interfaces with a Le Select server within a publishing site. A publishing site can be interfaced simultaneously with both data and service wrappers. On the other extreme of the architecture, a client application can invoke a given service that uses data from multiple publishing sites via a Le Select Client.

Wrappers manage metadata by providing a uniform representation of data, functions and programs with an extended relational model, and manage the execution of queries on local sources. The publishing mediator (Le Select Server) maps global queries into local queries, each for a different wrapper, and a composition query for producing the final result. It also has a runtime system to integrate the results of local queries. Global queries are expressed in an SQL-like language, that is, an SQL subset with specific extensions, which allows invoking functions or programs on data sources.

Publication sites can be organized as a hierarchy. Thus, a publication site can include a wrapper to a virtual database schema whose query-based specification can refer to information published by other publication sites. In this case, the schema corresponds to an integrated view of information published by other sites. The major advantage of this architecture is that the process of information publishing is completely decentralized via the publication sites.

Le Select's approach contrasts with previous information mediation systems such as Garlic, Disco, Himpar and Tsimmis, with respect to the integration policy. In these systems, publishing data at some site requires that a set of view definitions should be provided in some mediator located at another site. Their goal is to provide data transparency, which means hiding integration transformation details. When there are new data to be published, sometimes it is a difficult task for the publisher to reflect the changes into view definitions. Le Select does not automatically provide full transparency of data distribution because when building distributed SQL queries, a Le Select client references tables by their identifier, which contains the address of the publication site where the corresponding data have been published. However, the view definition service provided by Le Select enables the publication of virtual derived data, i.e., views. Hence, queries over the views hide the physical distribution of the underlying data from which the views are defined.

As a pioneer initiative on addressing data and programs heterogeneity and distribution management, Le Select did not adopt new coming standards like Web services and Grid computing. Instead, it relies on well-established open standards for interoperability. Network communication between Le Select components is assured via a CORBA protocol, although other means of communication are also possible. That is, JDBC statements between Le Select components (clients or servers) are embedded into CORBA/IIOP messages. However, Le Select is an isolated and proprietary initiative, which difficult its adoption by others.

3.1.2 Web Services

Web services are a new tendency that is going through a standardization process. The main concept behind Web services is the notion of publishing services over the Web, to be used by other programs. Web services relies on a service oriented architecture (SOA) (WS Architecture, 2002). SOA counts on the Web client/server infrastructure and uses a simpler and yet higher level protocol (Simple Object Access Protocol - SOAP). The idea is to allow different software applications, running on a variety of platforms to communicate. Through SOAP protocol, an application can have access to other applications' method invocation across the Internet, i.e., a method can be invoked remotely, and have its results delivered to the application that invoked it. Indeed, this feature addresses directly the problem of handling distributed program resources.

To become a Web service provider, there are two basic requirements: the ability to build and/or parse SOAP messages, and the ability to communicate over the network. Typically, a SOAP server (SOAP router) running together with a Web application server performs these functions. For instance, a Java class can have its methods "deployed" as services using the Apache AXIS engine (AXIS), which sends and processes SOAP messages. Alternatively, a programming language-specific runtime library can be used to provide them. In addition, each Web service provider publishes its interface as an XML document, using the Web Service Description Language (WSDL) (WSDL, 2003). This document specifies the service interface so that client applications can automatically bind to the Web service. After getting the binding information, the Web service requestor interacts with the provider by exchanging SOAP messages. Based on the WSDL document information, the requestor builds a SOAP message and sends it to the provider. Then, the provider receives the message, unwraps it, processes it, builds a SOAP message with the response, and sends it back to the requestor. Service provider and service requestor roles are logical constructs and a service may exhibit characteristics of both.

Legacy programs can also be encapsulated as Web services, but a Web service adapter has to be developed for each program to enable them as a service. Similarly to Le Select, to publish a legacy program as a Web service means to provide a wrapper for that legacy. In the case of the Web service architecture, the wrapper is a Web service adapter that is served by a Web service provider as shown in Figure 6.

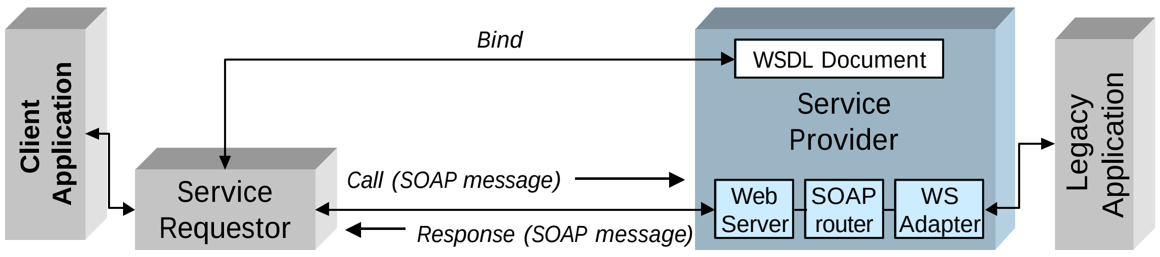


Figure 6: Web services Requestor/Provider Architecture

3.1.3 Grid Computing based Systems

Grid computing systems (GRID COMPUTING INFO CENTRE) focus on scaling up the computational power to process and access multi-Petabyte data. Typically, Grid computing based systems use the Globus infrastructure (GLOBUS) to distribute the execution of scientific programs. In Grid architectures, geographically distributed, heterogeneous collections of computing resources are accessed through a single point of contact. The Globus infrastructure has been proposed in the form of a toolkit (CZAJKOWSKI, K., FOSTER, I., et al., 1998) (FOSTER, I., KESSELMAN, C., 1999), which includes basic services to address Grid computing issues. The job execution service controls the submission and execution of jobs on remote machines. In the Globus toolkit, this service is called Globus Resource Allocation Management (GRAM).

GRAM service provides functionality that can be viewed as a three-tier architecture as shown in Figure 7. The client tier submits jobs to a remote resource. Each job is identified by a job ID that can be used to check on its status. Also it is possible to get a job status through event notification sent by the GRAM Server tier (middle tier). The middle tier consists of two basic elements: gatekeeper and job manager. The gatekeeper is responsible for the client authentication based on a previously defined security policy. Once the job request is approved, the GRAM server starts up a job manager for that request. From that moment on, the job manager will interact with the GRAM client. Finally, the backend tier is where the job actually runs.

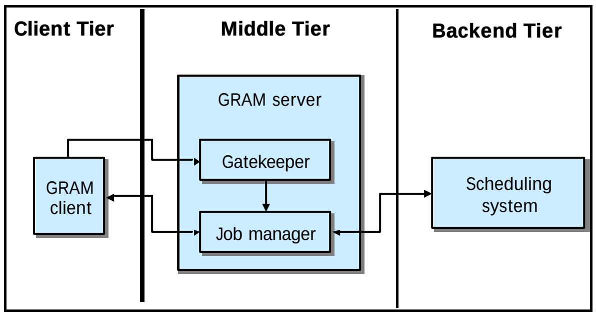


Figure 7: GRAM Architecture (adapted from FOSTER, I., KESSELMAN, C., *et al.,* 2002)

Many projects within Grid computing community are concerned with data retrieval (GRID DATAFARM PROJECT) (DATAGRID) (ALLCOCK, B., FOSTER, I., ET AL., 2001). These projects focus on managing access to multi-Petabyte distributed data. Chervenak et al. (CHERVENAK, A., FOSTER, I., ET AL., 2001) propose a two-layer Data Grid architecture. The first layer includes two basic services: storage systems and metadata management. The second layer includes higher-level services: replica management, replica selection and data filtering. The idea of having replicas is useful because some storage locations may offer better performance or availability for accesses to or from particular locations. For instance, the selection service may consider starting parallel connections to replicas, accessing complementary subsets of data. Particularly to scientific applications, most data resources requests are for read-only access, in which case, replica management becomes simpler. However, because the focus is on improving performance of data resources access, that is, offering more computational power, Grid based projects end up not addressing the data heterogeneity problem.

3.2 Describing scientific resources

Scientific resources management is not possible without describing them. Traditionally, the database community has contributed significantly to resources description, providing metadata solutions. Some definitions of metadata would simply define it as data that describes data (INMON, B., 1996). However, the increasing complexity of information systems requires a more sophisticated definition. Within the scientific community, where there is a high degree of resources heterogeneity and distribution, users need metadata not only for describing data but also for describing programs, models, workflows and experiments. A better definition is given by the Meta Data Coalition (MDC): "descriptive information about the structure and meaning of data and of the applications and processes that manipulate data" (MDC, 1999).

Large metadata needs management. Metadata management has gained increased importance especially because of the demand for integration of distributed and heterogeneous systems. There are two main approaches to address metadata management: metamodelling and ontology. The fundamental difference between these two approaches is on the attachment to the application. While metamodels are specifically built to support a metadata application, ontologies consist of relatively generic knowledge that can be reused by different kinds of applications (SPYNS, P, MEERSMAN, R., JARRAR, M., 2002).

Metadata models have been created within specific domains aiming to help interoperability. UDK model (GUENTHER, O.; VOISARD, A., 1997), for instance, is an agreed metadata model for describing data from environmental applications. These models have been helpful, but the inherent heterogeneity of scientific applications can not prevent new data models to emerge independently.

In a first initiative to establish standard data models, Microsoft and other partners have developed the Open Information Model (OIM), which was accepted as a standard by the MDC. Another initiative on the same direction emerged from IBM and other companies. They have developed the Common Warehouse Metadata (CWM), which was adopted by the Object Management Group (OMG) (OMG, 2000). The purpose of OIM and CWM initiatives was to support tool interoperability across technolo gies and companies via shared metadata, by providing a formal description of domain-specific metamodels. However, tools whose metamodels were not members of the set of domain-specific submodels could not interoperate.

To support the coexistence of different metamodels and facilitate the interoperability between applications that hold them, OMG has proposed the Meta-Object Facility (MOF, 1997). The MOF model corresponds to the most fundamental layer in a traditional 4-layer metamodelling architecture. This is a proven architecture for defining the precise semantics required by complex models that need to be reliably stored, shared, manipulated, and exchanged across tools. As it is used for defining metamodels, the MOF model is considered to be a meta-metamodel. A meta-metamodel can define multiple metamodels. To each metamodel there is at least one meta-metamodel associated (implicitly or explicitly), and then, a metamodel can be viewed as an instance of a meta-metamodel. Metamodels primary goal is to define a language for specifying models. A set of models that are instances of the same metamodel can easily share their metadata.

OMG officially adopted the Unified Modelling Language (UML) (UML REVISION TASK FORCE, 1999) as its object modelling standard (KOBRYN, C., 1999). UML is specified via a metamodel, derived from the meta-metamodel layer of the 4-layer architecture. In particular, the UML model is an instance of the MOF meta-metamodel.

Ontologies have been used as second approach, to address the metadata management problem. Analogously to the meta-metamodel (MOF) approach, domain ontologies are conceived to address a set of applications that need to interoperate on the metadata level. A domain ontology expresses a community's consensus knowledge within a given domain. A computer ontology is defined as an "agreement about a shared, formal, explicit and partial account of a conceptualization" (GUARINO, N., GIARETTA, P., 1995) (USHOLD, M., KING, M., 1995). Each domain ontology contains the vocabulary of terms and the definition of concepts and their relationships for that domain. In many ontology based applications, the instances (domain resources) are included in the ontology (SPYNS, P, MEERSMAN, R., JARRAR, M., 2002), together with the concepts and their relationships.

Some domain-specific ontologies, such as the Gene Ontology (ASHBURNER M., LEWIS S., 2002), have been proposed. However, neither a standard metamodel has yet been universally accepted, nor a generic scientific resources ontology has yet been proposed. Without such representation formalism, independent initiatives have been developed to address scientific resources management. Although none of these approaches are definitive, they contribute for a future efficient metadata management of scientific resources. We have selected some of the most relevant contributions to scientific resources metamodelling, which are discussed in the next sub-sections.

In section 2.3 we have evidenced programs and models as distinct resources. This distinction is particularly important because it allows scientists to retrieve and consider similar programs as alternatives to solve the same problem. Some of the approaches described here provide ways for finding program resources and learning how to use them correctly. Initially, we present Web services' (section 3.2.1) and Grid's (section 3.2.2) support for describing the program resources they publish. Then, on the following sections we present the most representative projects that focus on metadata issues for scientific resources. A final discussion is left to the last section, where we point out that most of them do not consider programs and models as distinct resources, and consequently find difficulties in their management.

3.2.1 WSA

More than to address communication between program components, the Web services architecture (WSA) (WS Architecture, 2002) also addresses program description and program search. As shown in Figure 8, the complete Web service architecture comprises three roles: requestor, provider and registry. The third role acts as a service discovery agency, where the service provider publishes (step 1) its service description. Then, the service requestor uses a find (step 2) operation available at the service discovery agency to retrieve the desired service. Then the service requestor uses the service description to bind (step 3) with the service provider and invokes (step 4) the Web service implementation.

A Web service may be published at multiple service registries. Each service provider has its own registry service, thus allowing for direct bindings with the service requestors. Direct binding usually occurs between two business partners that have previously agreed on terms of doing e-business over the Web. Typically, a requestor does not know where to find available services. Therefore, independent services registry servers are needed. There may be several types of registries, which are characterized according to their restrictions to specific domain communities, such as, the internal enterprise departments, the set of external partners of a company (company portal registry), the business public (generic registry), among others. The Universal Description, Discovery and Integration (UDDI) (UDDI) specification for Web service registry was proposed by an industry consortium lead by IBM, Microsoft and Ariba. Another registry specification is the ebXML Registry (EBXML), which has been proposed by OASIS. Both propose a data model for registering, storing and finding Web services description documents (WSDL documents).

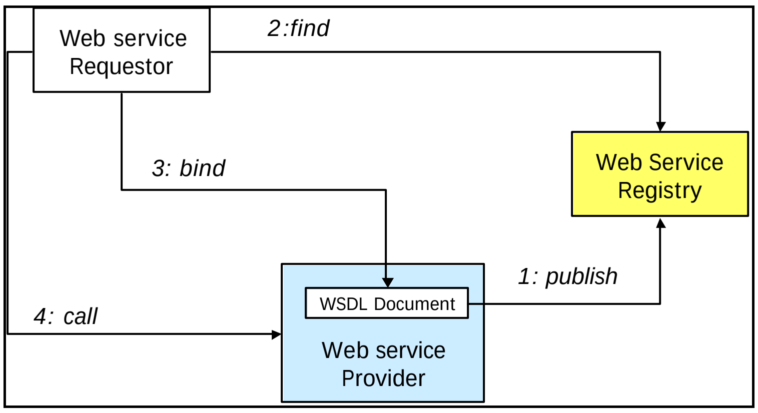


Figure 8: Web services Architecture

While program search is provided through the Web service registry, program description is provided through WSDL documents. A WSDL document contains a set of inter-related specifications to describe services, such as `messages`, `ports`, `port types`, and `bindings`, presented in Figure 9. These specifications can be classified as abstract and concrete. The abstract specifications describe the program abstract interface (`port type`), and its input and output `messages`. The concrete specifications actually extend the abstract descriptions to describe how to access the real code (service instance). The concrete protocol and data format specification for a particular `port type` defines a reusable `binding`. Associating a network address to an existing `binding` specifies a `port`. Finally, a `service` is defined as a collection of network endpoints or `ports`. This separation of abstract and concrete definitions allows abstract definitions to be reused, i.e., the same program can be served at different network locations, while sharing the same abstract description.

A WSDL document begins with a root tag called definitions. Figure 9 shows a simplified view of the WSDL schema, where the main elements are defined under the definitions element. The `portType` element is described by a set of `operation` elements. Each `portType operation` has exactly one input and one output relationship. Each one of these relationships refers to one `message` element instance. A `message` element is described by a set of `part` elements, and each `part` element must refer to a type description. WSDL does not introduce a new type definition language. Instead, it supports XML Schema (XML Schema) as its type system. Thus, each `part` element refers to a basic XML Schema data type or to an XML Schema element or complex type previously defined under the `types` element. Nevertheless, the use of XML Schema type system is not obligatory, as WSDL allows the use of other type definition languages, through its extensibility mechanism.

At the concrete level, a binding to specific message encodings and protocols are assigned to the abstract definitions. The `binding` element refers to a specific `portType` element, through its `type` attribute. The `binding` element is described by a set of `bindingOperation` elements. For each `bindingOperation` element there is a correspondent `portType operation` element. The `service` element is described by a set of `port` elements, and each `port` element refers to a specific `binding` element. In addition to its vocabulary, WSDL also allows the use of specific binding extensions to support the specification of protocols and message encodings. Therefore, other vocabulary tags are used inside the `binding` element definition, to define, for instance, the style and transport protocol used during the message exchange (e.g. RPC style, SOAP protocol).

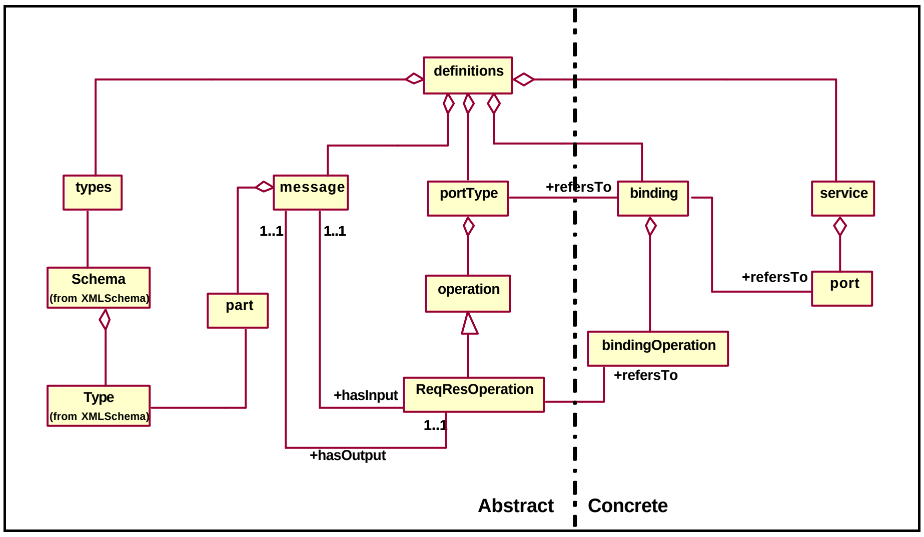


Figure 9: WSDL schema (simplified)

Figure 10 shows a simplified fraction of the WSDL XML schema (WSDL Schema) where the main elements appear under the group named `anyTopLevelOptionalElement`. The `any` tag is a special XML element that allows for the insertion of literally any element into the XML document instance. According to this schema, any extensibility element is allowed under the definitions element. Therefore, we can create other description elements that may be included in a WSDL document through the `import` element.

```
<element name="definitions" type="wsdl:tDefinitions" />
<complexType name="tDefinitions">
    <complexContent>
        <extension base="wsdl:tExtensibleDocumented">
            <sequence>
            <group ref="wsdl:anyTopLevelOptionalElement" minOccurs="0"
                  maxOccurs="unbounded"/>
            </sequence>
            <attribute name="targetNamespace" type="anyURI" use="optional"/>
            <attribute name="name" type="NCName" use="optional"/>
        </extension>
    </complexContent>
</complexType>
<complexType name="tExtensibleDocumented" abstract="true">
    <complexContent>
        <extension base="wsdl:tDocumented">
            <sequence>
            <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<group name="anyTopLevelOptionalElement">
    <choice>
        <element name="import" type="wsdl:tImport"/>
        <element name="types" type="wsdl:tTypes"/>
        <element name="message" type="wsdl:tMessage" />
        <element name="portType" type="wsdl:tPortType"/>
        <element name="binding" type="wsdl:tBinding"/>
        <element name="service" type="wsdl:tService"/>
    </choice>
</group>
```

Figure 10: Some WSDL schema global elements (simplified)

Web services are on the right track to add semantic to Web resources. Although they are now limited to program description, they can be extended to provide higher-level descriptions, such as model descriptions. Web services registries data models intend to provide solutions towards this, acting as a WSDL complement, and adding higher-level description elements to the ones presented in Figure 9. However, WSDL alone will not solve semantic interoperability issues. They provide a uniform access to a metamodel management, but ontologies and metadata on a specific domain are still needed.

3.2.2 OGSA

A recent work (FOSTER, I.; KESSELMAN, C.; NICK, J. M.; TUECKE, S., 2002) from Globus Project is bringing Grid and Web services technologies together through the Open Grid Services Architecture (OGSA). This is a workgroup proposition still under discussion within the Global Grid Forum (GGF), which has draft updates coming up frequently (almost monthly). The Open Grid Service Infrastructure (OGSI) document (OGSI-WG, 2003) presents a draft with a full specification of the behaviours and WSDL interfaces that define a Grid service. A just released API implementation is also available for use, the Globus Toolkit 3.0 (GT3).

OGSA proposes to represent every Grid service as a Web service that conforms to a set of conventions and supports standard interfaces. These conventions and interfaces are useful for building higher-level service descriptions. Grid service interfaces correspond to `portTypes` in WSDL, as shown in Figure 11. According to OGSA, all but one of these interfaces are optional. The required one is the `GridService` portType, which is responsible for the user control over the service state. There are other optional portTypes that may be optionally available, such as the `Factory`, responsible for the service creation, the `Registry`, responsible for the service registration, and the `Notification-sink` and `Notification-source`, to provide notification facilities.

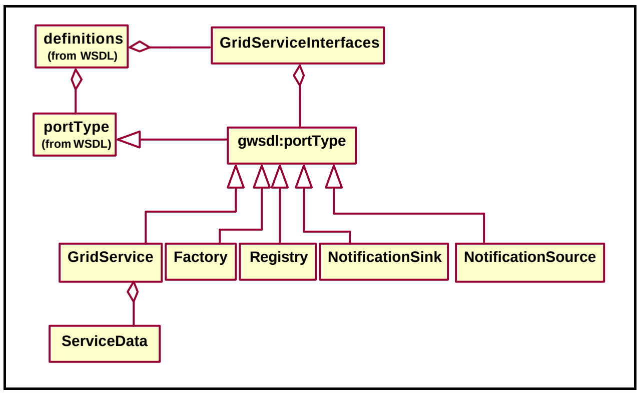


Figure 11: OGSI WSDL extension

The main idea is to capture service semantic with respect to the service instance's state information to service requestors for query and change notification. The term used is `ServiceData`, whose elements are accessible through operations defined at the `GridService` porttype. The `GridService` portType includes three basic operations: `FindServiceData`, `Destroy` and `SetTerminationType`. The first one returns information about a service's state, execution environment and additional semantic details. The `destroy` operation allows an authorized client to kill the executing instance of a service. Finally, the `SetTerminationTime` operation extends the lifetime of a service, as it is usually associated to an expiration date.

An OGSI extended WSDL document includes new `portType` elements that are extensions of the WSDL portType element. Basically, the extension involves the inclusion of a new `portType` child element named `serviceData`, used to define `serviceData` elements, associated with that portType. Optionally, initial values for those `serviceData` elements (marked as "static" `serviceData` elements) may be specified using the `staticServiceDataValues` element within portType. The example in Figure 12 shows a GridService portType, which is not the original WSDL `portType` element, as it is under the `gwsdl` namespace, which includes `serviceData` elements.

```
<wsdl:definitions xmlns:tns=”xxx” targetNamespace=”xxx”>
 <types >
   <xsd:schema … >
       <xsd:complexType name=”someComplexType”>
            ....
   </xsd:schema >
  </types>
   …
  <gwsdl:portType name=”GridService”>
    <wsdl:operation name =”NCName”>
     …
   <sd:serviceData name=”sd1” type=”xsd:String” mutability=”static”/>
   <sd:serviceData name=”sd2” type=”tns:someComplexType” />
   <sd:serviceData name=”factoryHandle” type=”ogsi:HandleType”
                  minOccurs=”1” mutability =”constant” nillable=”true”/>
   <sd:serviceData name=”gridServiceHandle” type=”ogsi:HandleType”
                  minOccurs=”0” maxOccurs=”unbounded” mutability=”extendable”/>
  …
   <sd:staticServiceDataValues>
      <tns:sd1>initValue</tns:sd1>     …
   </sd:staticServiceDataValues>
   …
  </gwsdl:portType>
</wsdl:definitions>
```

Figure 12: Extending WSDL with OGSI elements example (adapted from OGSI-WG, 2003)

Through available WSDL documents (including the standard portTypes mentioned), Grid clients are able to access Grid service descriptions and dynamically discover, register and compose Grid services. However, although Grid service descriptions are not limited to what Web services can provide, so far the Grid community focus has been on service execution information, and not on higher-level service information, such as information about the service business area.

3.2.3 MyGrid Project

MyGrid (MYGRID PROJECT) is a research project that is based on the Grid framework, focusing on bioinformatics applications. A recent work (WROE, C. *et al.*, 2003) describes in more detail the Grid system under development, highlighting that it provides a variety of supplementary services to OGSA-based systems.

MyGrid architecture is shown in Figure 13. The client framework provides user access to MyGrid server functionality, through a Web portal, which includes repository, workflow and ontology clients. The server includes workflow management facilities, which have access to Bioinformatics programs. The workflow repository stores workflow specifications, and the enactment engine is responsible for workflow instantiations and actual executions. MyGrid assumes that Bioinformatics programs are encapsulated as Web Services. Scientists can then run those programs through Web services requests. The personal repository acts like a laboratory electronic log, where scientists' personal data and provenance information is stored. Each Web service execution is logged into the personal repository.

The description and classification of bioinformatics resources (services, data and workflows) are provided through metadata services and directories. MyGrid classifies metadata into two broad categories: domain and business metadata. Domain metadata refer to service classification and abstract service input/output data types (e.g., BLASTn is a tool for computing sequence homology that uses the BLAST algorithm over nucleotides). Business metadata refer to specific serviced code resource, its location, reliability, and version (e.g., BLASTn service offered by EBI is 80% reliable), as well as information about its usage (e.g., date, time, particular parameter values when a BLASTn was actually enacted). In fact, they organize these metadata into four layers: class of service, abstract service, instance service and invoked instance service. This four layer description helps in finding alternative services, i.e., instance services can be considered to be valid candidates when they belong to the same service class or abstract description. The workflow instantiation module can benefit from this layered service descriptions as it processes abstract workflow specifications. Available candidate instance services are provided by the service type directory.

MyGrid uses an ontology-based approach, providing a suite of ontologies expressed in DAML+OIL (HORROCKS, I., 2002). This suite comprises a set of inter-related ontologies, where each ontology provides a vocabulary of terms or concepts, and their inter-relationships to form resource descriptions of a specific domain. DAML+OIL describes each domain in terms of classes and properties, and its formal approach provides reasoning facilities. This reasoning is particularly useful to support the suite of ontologies management and to support resource description. For example, after describing BLASTn and BLASTp code resources, and associating them to the bioinformatics ontology concepts, the DAML+OIL reasoner automatically generates a classification hierarchy of these resources. Then, the user can access the ontology client to browse the layers of metadata available (of the bioinformatics ontology) and find a list of the equivalent available resources (service instances).

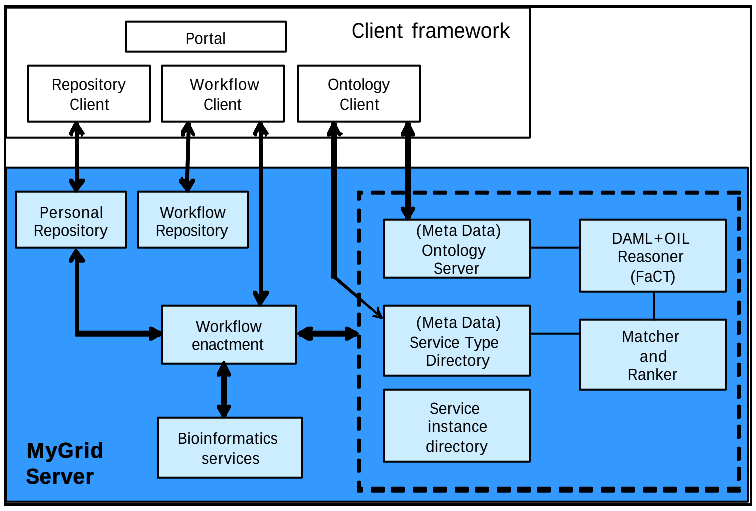


Figure 13: MyGrid Architecture (WROE, C. *et al.*, 2003)

MyGrid adopts a four-tiered model to describe services: class of service; abstract service; instance service; and invoked instance. According to this classification, MyGrid description does not consider the description of service implementations of a given abstract service. The instance service description corresponds to the available compilation, ready to be executed, while the abstract service description corresponds to the program abstraction, e.g., its algorithm. However, between these two concepts we believe there should be an extra layer to represent the service implementation.

As MyGrid is based on an ontology approach, there is no metamodel or conceptual diagram to analyze. Instead, they organize concepts and instances as a suite of ontologies presented in Figure 14. The Web services ontology uses some of the concepts included in DAML-S ontology (ANKOLEKAR, A. *et al.*, 2001) and extends it with some specific concepts to address the bioinformatics domain. One of the main concepts in DAML-S is the service profile, which describes a service using properties like name, purpose, function, etc. MyGrid proposes extensions that include the addition of the `uses_method` property, to describe to which method or algorithm a service should be associated. This concept is used to connect service descriptions to the bioinformatics ontology. For instance, service `Blast-p_service uses_method Blast_algorithm`. Although this concept allows associating a program (Blast-p) to its algorithm (Blast), it works as a simple classification. The idea of describing the algorithm itself as a scientific model with its own characteristics is not considered. Therefore, we may say that MyGrid approach does not explicitly represent and manage scientific models as an independent concept. Also the relationship between programs, data and models are rather clumsy, since they are not explicitly represented. These concepts and their instances are all mixed inside the same ontology. Finally, MyGrid suite of ontologies was specially created to attend bioinformatics applications, but its organization does not seem to be easy to generalize. It would be useful to have a suite of ontologies designed to address other specific domain applications, in such a way that it would be necessary just to have one or two ontologies substituted.

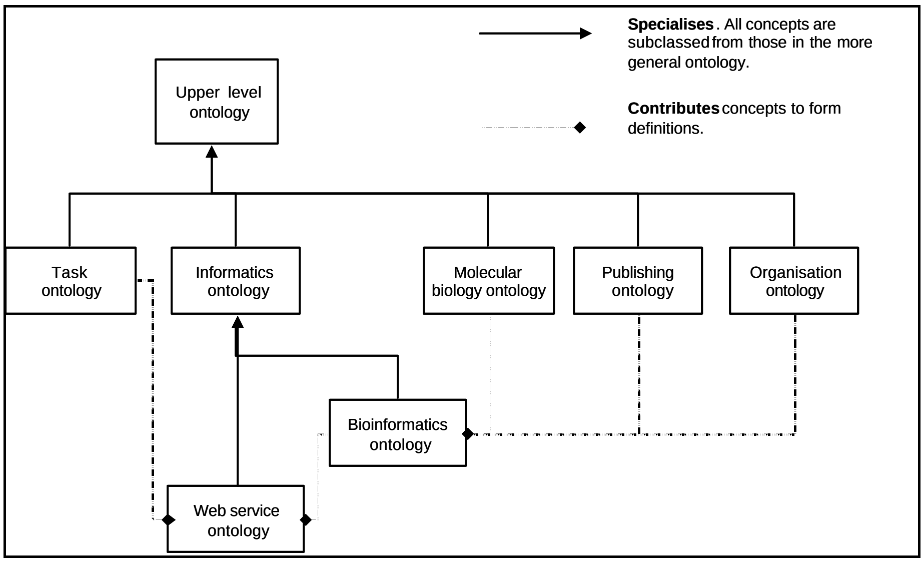

Specialises. All concepts are subclassed from those in the more general ontology.
Contributes concepts to form definitions.
Upper level ontology
Task ontology
Informatics ontology
Molecular biology ontology
Publishing ontology
Organisation ontology
Bioinformatics ontology
Web service ontology

Figure 14: MyGrid suite of ontologies (WROE, C. *et al*., 2003)

MyGrid works hard on attending many requirements of bioinformatics applications. Although it uses an ontology-based approach, it actually uses the Web services ontology as its metamodel for describing bioinformatics services. A Web application was developed over this metamodel, and its user interface guides the user on describing services, counting on the associated ontologies to build specific drop-down value lists. However, when trying to address other domains, MyGrid approach is not very flexible. Specific-domain ontologies should be built, and in the absence of practical methods to build ontologies, this is not an easy task. Also, there are not many available ontologies that are accepted by scientific communities.

One of the main purposes of an ontology is to provide a generic and common terminology classification on a specific domain. However domain specialists find it hard to agree on this common framework. An example is the Gene Ontology initiative (ASHBURNER M., LEWIS S., 2002). Despite its efforts on being freely available and using portable technology such as XML, it is not widely accepted and MyGrid choose to use build its own ontology. Therefore, to count on existing ontologies is the main weakness of MyGrid approach.

3.2.4 GriPhyN Project

The GriPhyN Project, another important project on data lineage, has developed the Chimera System (FOSTER, I.; VOECKLER, J.; WILDE, M.; ZHAO, Y., 2002). The architecture presented for Chimera is based on the metamodel approach and presents the Chimera Virtual Data Schema, which provides a representation of the computational procedures used to derive data. It defines a set of concepts and relationships that are used to capture and formalize descriptions of how a program can be invoked, and to record its potential and/or actual invocations.

As shown in Figure 15, a Chimera Client interacts with Chimera through a Virtual Data Language (VDL), for both data definition and data retrieval statements. The VDL Interpreter is responsible for translating VDL commands into SQL commands, which are issued to the Virtual Data Catalog (VDC). The VDC database implements the Chimera Virtual Data Schema, and actually stores information about data derivation. The Virtual Data Browser and Planner are examples of Chimera clients, which use the VDL to explore VDC contents and to develop plans for computations, respectively.

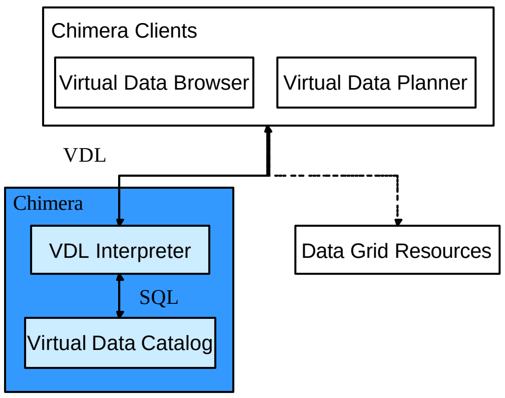


Figure 15: Chimera System Architecture (adapted from FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003)

According to (FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003), VDC information may be distributed across multiple locations, and hyperlinked. The idea is to identify each resource by a URL, allowing inter-catalog references.

Another interesting aspect of Chimera is its possible integration with Data Grids. Some Chimera clients were developed to provide such integration. A Chimera client can, for example, provide for Grid execution planning, based on VDC´s transformation instances. It receives an abstract workflow description from Chimera, produces a concrete workflow, and submits it to meta-scheduler for execution.

The Chimera Virtual Data Schema defines the concepts and relationships of the Chimera metamodel, which are presented in Figure 16. The `dataset` and `replica` concepts describe data resources, while the `transformation` concept describes the programs that transform data. Each `dataset` has a `type`, which specifies the various characteristics of the data set, including its storage structure or representation, data server information and what kind of data it contains. A `dataset` may have multiple `replicas` at different locations.

A `transformation` is a program interface that may take `datasets` of a given `type`, by reference. Such association occurs when the user starts a `derivation` that specifies which `datasets` will match which `transformation` input `type`. The `invocation` concept completes the execution information by describing the physical environment and time of the real execution (e.g., date, time, processor, operational system, etc.). Note that an invocation may be associated to a `replica` of the `dataset` associated to the `derivation`. The `replica` concept is important especially to data Grid environments, where performance for data processing is achieved through multiple parallel `invocations` of a `derivation`.

Foster et al. (FOSTER, I.; VOECKLER, J.; WILDE, M.; ZHAO, Y., 2002) highlight the possibility of a Chimera client to develop plans for computations, i.e., workflow specifications. A more recent work (FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003) states that there are compound and simple transformations. A `compound transformation` is composed by one or more transformations in a direct acyclic graph (DAG). However, so far, this concept has not been explicitly reflected in their schema.

Analysing the Chimera Virtual Schema more carefully, we identify some points of discussion. The `type` concept is representing two distinct concepts: the `transformation` input/output data type and the `dataset` type. Consider, for instance that a transformation may have two data inputs of the same `type`. It would be necessary to instantiate two equal `types`, to represent both data inputs. More than having this redundant description, it may cause some conflicts. When describing a `dataset` that would fit both `types`, which `dataset` would be a better choice? Yet, when describing a `derivation,` which `datasets` will be associated to which `transformation` input type? Do their `types` have to match? Separating dataset and transformation input/output types, as two different concepts, may solve such problem. The last one, should make a reference to the first one, i.e., a `transformation` input/output `type` should refer to a `dataset` `type`.

In the first version of their schema there was a clear separation of logical and physical transformation concepts. We took the liberty of connecting the schemas presented in both works to discuss them a little further. Please refer to those works to see the original schema figures. The `physical transformation` corresponds to the code resource that fits into a `logical transformation` description. A `physical transformation` is related to exactly one `logical transformation`, while a `logical transformation` may have many `physical transformations`. Even though the relationship is not present in the Chimera initial schema, we consider that an `invocation` should refer to exactly one `physical transformation` (dashed line in Figure 16), describing which physical transformation was used at the time of the execution. This relationship is particularly important to address provenance in an environment where code resources can be replicated, such as Grid computing environments. Finally, it is important to notice that the logical transformation concept is not representing the model or algorithm associated to a source code.

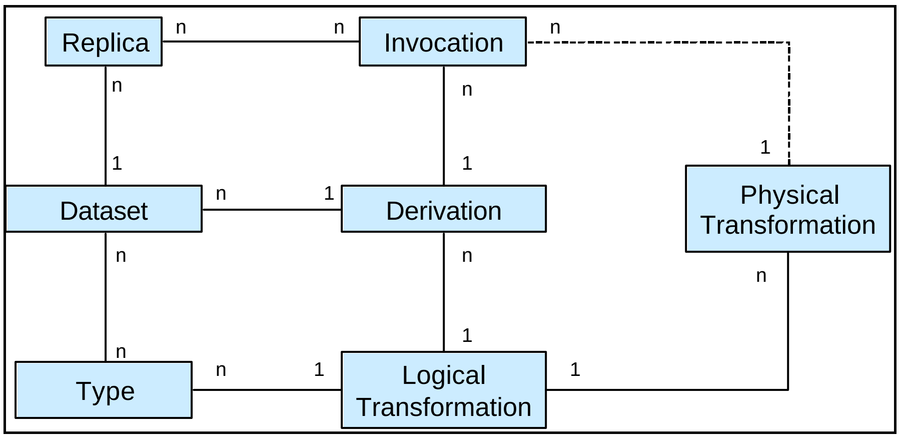


Figure 16: Chimera Virtual Data Schema

This project is on the right track with respect to scientific resources description, however, its metamodel needs some further developments to properly address more abstract concepts such as scientific models.

3.2.5 ESSW Project

The Earth System Science Workbench project (ESSW PROJECT) proposes a data management infrastructure for researchers who desire to publish large data sets derived from environmental models executions and global satellite imagery. They provide a framework for defining and collecting metadata for Earth science and environmental models. ESSW takes a metamodel-based approach, where the main concept is the science object, which is used for defining and collecting metadata for those models. Essentially, a science object is an entity that represents real world scientific resources such as models, their inputs and outputs, experiments (model executions) and experiment steps.

As shown in Figure 17, ESSW architecture consists of two main components: The Lab Notebook (LN) and the Labware. The Lab Notebook is the digital analogy to the handwritten laboratory notes of a scientist. It is a client/server application that logs metadata and lineage for scientific experiments and their related science objects as XML documents stored in a relational database. These metadata assists researchers or others with identifying particular data products, as well as tracking the steps leading to creation of a product, or the data lineage.

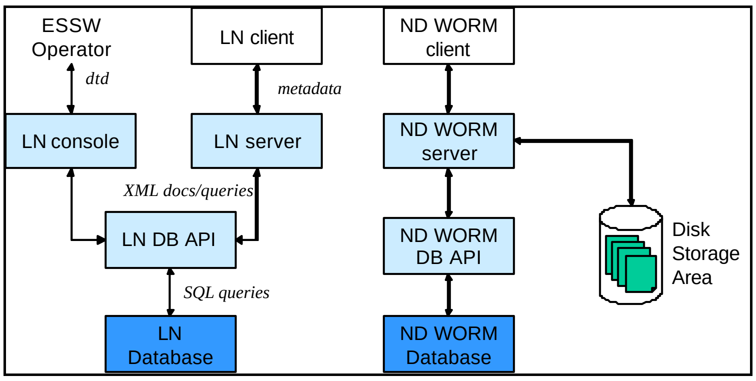


Figure 17: ESSW Architecture (adapted from FREW, J.; BOSE, R., 2001)

The LN client sends metadata as XML documents that are parsed, manipulated and validated by the LN server. These documents are stored in the LN database. The LN DB API is responsible for transforming XML documents into table tuples, and for translating XML queries into SQL queries. The console provides an interface for submitting XML DTDs to the lab Notebook, and creating metadata templates for science objects. The LN database contains DTD libraries and templates, and science objects. There are LN client tools to search, order and manage the LN database. A set of graph drawing tools generate directed graphs of experiment workflows. The LN lineage tool can display the metadata for any science object shown by clicking on it.

Labware is the digital analogy to the collection of equipment and instruments in a scientist's laboratory. It includes No Duplicate-Write Once Read Many (ND-WORM) services, which provide robust file archiving. The ND-WORM is a client/server application that maintains metadata about system files. It acts as a data resource catalogue. The process of cataloguing files involves assigning each file a unique id, which is calculated when the file is copied into the disk storage area. Metadata information about these files also includes the original location of the files, file hierarchies, and keywords. ND-WORM can provide search tools for retrieving files or sets of files catalogued under the same keywords or hierarchies. Experiment data outputs are also catalogued through ND-WORM facilities.

The `science object` is at the core of the ESSW metamodel. Essentially, science objects represent real world scientific items such as files and data processing routines. Other important concepts are associated to the science object, such as `experiment` and `model`. An `experiment` is defined as the execution of some `model`. Each experiment consists of one or more `experiment steps`, which represents a computational process with `inputs` and `outputs`. However, each `experiment` is related to only one `model`, meaning these steps correspond to different executions of the same model. The relationship between two science objects is represented by the `ScienceObjectLink` table. This relationship allows for the registry of the lineage for an `experiment`, linking two subsequent executions.

The ESSW metamodel is extensible to represent user defined science objects. When metadata templates that define new science objects (inputs/outputs) for a model are needed, a new table is created for the new type of science object. The scientist is free to provide his own metadata templates in the form of XML DTD's. The new table is named after the DTD root. The new table structure includes some attributes inherited from the science object table, and some attributes reflecting the DTD structure (XML tags).

Since the ESSW metamodel is focused on tracking experiment lineage, in its metamodel, a `model` publication occurs in the context of documenting an `experiment`. Actually, the `model` concept represents an executable code. As we pointed out before, in the scientific community domain, there are three concepts that should be explicitly represented: the model itself, its implementation (the program) and the code that is actually executed (compilation). In the ESSW metamodel, these three concepts are represented in one concept. This concept overload does not help the management of scientific resources.

Another point of discussion is how ESSW registers scientific experiments. The scientific community needs tools for annotating multiple models or programs experiments. The ESSW metamodel is now limited to register single model executions and how they are linked to each other, being able to retrieve workflow instances. However, since the experiment concept is associated to a single model execution, there is not an explicit concept to represent experiments that involve a set of model executions. Moreover, although workflow instances are implicitly registered, it is worth mentioning that the workflow specification is not covered by the ESSW metamodel.

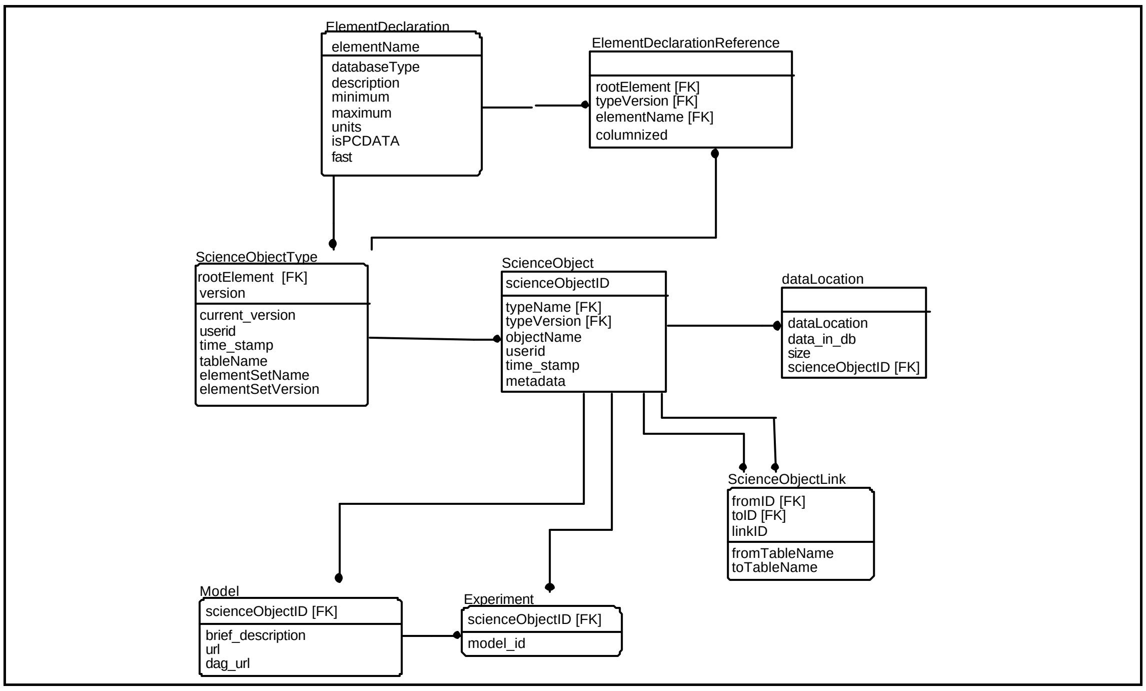


Figure 18: LN Database Schema (FREW, J.; BOSE, R., 2001)

Despite being extensible, the ESSW metamodel needs further enhancements to address scientific resources descriptions. Particularly, it lacks adequate abstract definitions for scientific programs and models, which would provide a richer scientific resources description and enable workflow definitions.

3.2.6 ESP2Net Project

Similar to ESSW, the Earth Science Partners' Private Network project (ESP2NET PROJECT) focus is on interchanging scientific datasets and publishing the experiments that generated them. They propose an active semi-structured information sharing system architecture (ASSISS) that combines several complementary means of sharing the experiences from scientific experiments: browsing, searching, and active dissemination. Within this architecture, they adopt a metamodel-based approach. Although we could not find an explicit presentation of their metamodel, in the work (KAESTLE, G., SHEK, E.C., DAO, S. K., 1999) the authors propose its representation as a Scientific Experiment Markup Language (SEML) based on XML to capture scientific experiments.

ASSISS interfaces with distributed scientific information repositories and services for transparent distributed access and processing of large scientific datasets. As experiments are being conducted, SEML documents capturing the experiments can be proactively disseminated by means of reliable multicast to groups of users interested in some aspect of the activity, as shown in Figure 19.

Browsing is achieved by extended Internet browsers with the ability to display SEML documents. All SEML documents are stored as BLOBs into a relational DBMS, and descriptive attributes are extracted for indexing and faster query. ASSISS also provides a harvester that harvests SEML repositories and their documents, and builds a master index that allows scientists to manage experiments locally, but to search for experiments nationally. SEML provides a Web-based interface to browse, search and mine a SEML repository.

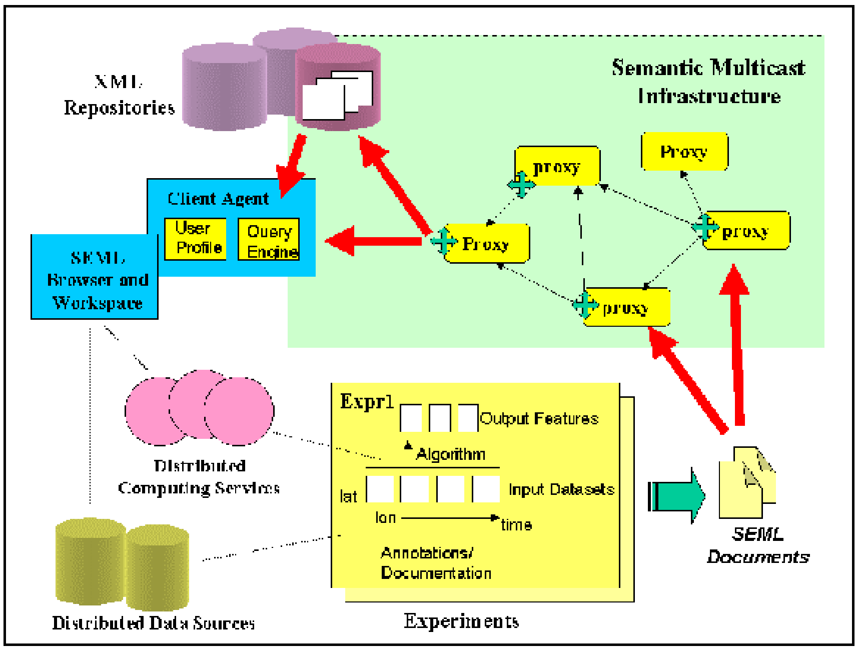


Figure 19: ASSISS Architecture (ESP2NET PROJECT)

SEML is implemented as a Document Type Definition (DTD). SEML DTD is said to be under continuous evolution to fit the changing needs of the scientific community. However, a more recent version of it could not be found. SEML attempts to explicitly capture the experiment process where datasets are involved. A collection of SEML documents can be viewed as an electronic experiment log, which can be useful for future reuse or tuning.

According to the SEML DTD, a SEML document may be divided into three parts: identification, documentation and experiment, as shown in Figure 20. The identification part includes information about the people and organizations involved in the experiment, for further contact. The documentation part contains information about the experiment as a whole, as well as specific information, embedded in a particular element. Finally, the experiment part includes information about all the `processes` involved in the `experiment`, as well as their `inputs` and `outputs`. Each `input`, `output` or `process` may be associated to elements at the documentation part, such as an `annotation` or a `resource` element. The `resource` element contains a link to a resource over the network. A `process` element may be simply a document describing what was done to transform a certain `input` into a certain `output`; or a distributed computing service that is actually invoked to transform a certain `input` into `output` results. To allow references between SEML document elements, the referenced items must have unique identifiers.

SEML documents were designed to document past experiments, however they can be viewed as templates to further similar experiments. Although the whole experiment may be reused, SEML DTD does not provide reusability for the internal elements. All the concepts representing scientific resources are defined within an experiment. In the scientific community, resources like processes and data are used independently for the composition of different types of experiments. As SEML documents do not provide support for reusing these resources, their descriptions have to be replicated in every document that needs to include them.

Another point of discussion is related to the process concept. This concept represents an executable code. Its description may exist implicitly, as an annotation within the document that uses it. Therefore, SEML structure does not include elements to explicitly describe models and/or programs related to it. Analogously, data input and output are also concepts related to real data resources, meaning their description is also implicitly and optionally available within annotation elements.

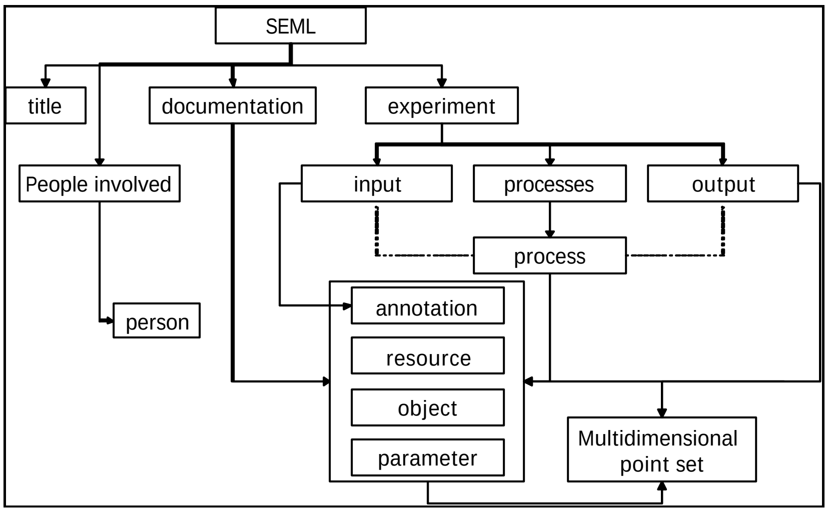


Figure 20: SEML Structure (KAESTLE, G., SHEK, E.C., DAO, S. K., 1999)

ESP2Net language needs further enhancements to address scientific resources descriptions, and a metamodel design would help. Similarly to ESSW metamodel enhancement suggestions, it would be useful to provide abstract definitions for scientific programs and models, providing a richer scientific resources description and enabling workflow definitions.

3.2.7 ECOBAS-MIF Project

The ECOBAS Model Interchange Format (ECOBAS_MIF) (BENZ, J; HOCH, R., 1999) (GABELE, T.; BENZ, J.; HOCH, R., 1999) is a project of the International Society for Ecological Modelling (ISEM-Europe). It was developed to support the exchange of models between various System Dynamics simulators. In the MIF format, for a complete documentation of a model, the characterization of the ecological environment for which the mathematical model has been created and validated is provided. Although MIF format is presented in the form of a structured list of descriptors, the ECOBAS-MIF can be viewed as a metamodel-based approach.

In an attempt to solve the problem of model documentation and retrieval, Benz *et al.* (BENZ, J.; HOCH, R.; GABELE, T., 1997), produced the REM-ECOBAS system. It organizes model documentation in two levels of information. REM, the Register of Ecological Models, is in charge of the first level, which includes metainformation about models, such as a contact address of the author(s), abstract, references, Internet based links (URL's) to sources of information and comments by model developers or users. ECOBAS is in charge of the second level, which provides detailed and complete descriptions of the mathematics of each model.

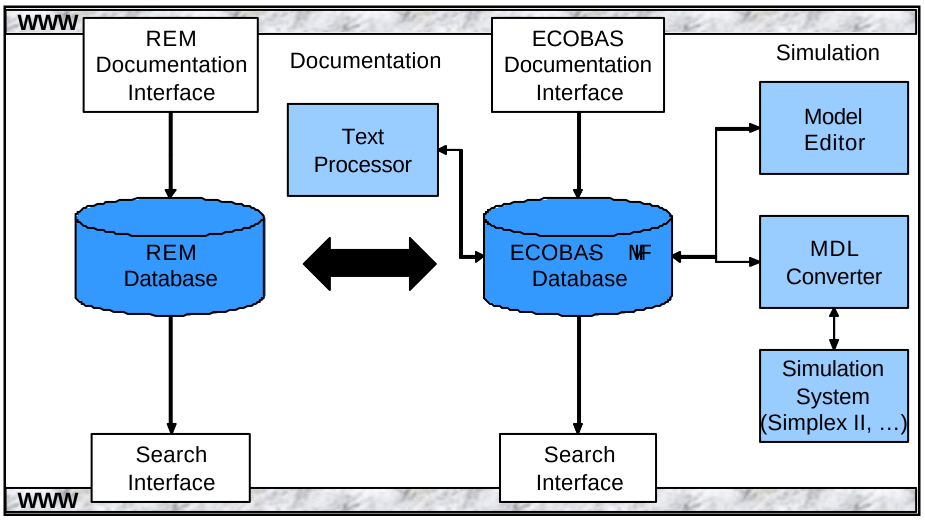


Figure 21: REM-ECOBAS Systems Architecture

An overview of ECOBAS architecture is given in Figure 21. In REM and ECOBAS systems, a search engine is available at the Web, to find model documentation. A documentation interface is also available at the Web, where users can describe their models. The ECOBAS documentation interface generates an ECOBAS-

MIF format file, which is stored in the database. An important aspect of ECOBAS system is the coupling of documentation and source code. Once some model documentation is completed, it can be used to generate text documents linked with the source code of the corresponding simulation model. For that purpose, a text processor that translates an ECOBAS-MIF file into a TEX file format has been implemented. Other file conversion options are under development, which includes a model generator, a graphical model editor and converters to integrated simulation system environments.

REM database currently stores 647 models (REM). For each model a so called info-sheet is created and stored in the REM database, according to the structure shown in Figure 22.

Section	**field/category**	**referring to other (more detailed) information**
1. General Model Information		
	Name	
	Acronym	
	Main medium	
	Main subject	
	Organization level	
	Type of model	
	Keywords	
	Contact	+
	Author(s)	
	Abstract	
II. Technical Information		
II.1 Executables		+
II.2 Source-code		+
II.3 Manuals		+
II.4 Data		+
III. Mathematical Information		+ (ECOBAS)
III.1 Mathematics		
III.2 Quantities		
III.2.1 Input		
III.2.2 Output		
IV. References		+
V. Further information in the World-Wide-Web		+
VI. Additional remarks		

Figure 22: REM database structure (REM)

The ECOBAS-MIF consists of a set of elements that describes a model. These elements are organized in three main sections: Type, Specification and Domain sections as shown in Figure 23. The `Type` section is responsible for information about model abstract structure, e.g., its components, procedures and functions. The

Specification section contains more specific information about the model, such as parameter values and ranges. The Domain section includes information about the application area to which the model is used for.

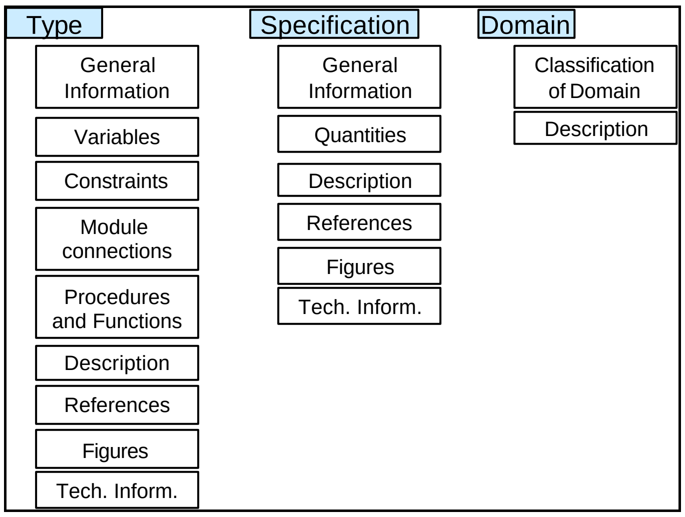


Figure 23: ECOBAS_MIF Structure

The General information section contains information about model identification, e.g., model name, authors, reviewers, keywords and application domain. The process simulation type is also declared in this section. It provides an overall model classification according to time dependency (event, static, discrete, continuous and aggregate). The Variables element is used to declare a list of: constants, state variables, time variables, dependent variables, input variables and space variables. It is important to notice that the type section includes information about model internal procedures and functions, characterizing a white box description approach. Components must be declared in the Module connection section when the simulation process type is an aggregate. The Description section is used to explain the purpose of the model, hypotheses and equations in free text form. The References section is used to include bibliographic references, and the Technical information section is used to declare

The Specification section includes a Quantities subsection, which is used to declare numerical default values and allowed ranges for the variables declared in the Type section.

```
Type Section
GENERAL INFORMATION:
NAME: nitrification-in-water;
AUTHOR: J. Benz; benz@wiz.uni-kassel.de;
MODEL: CERES-WHEAT;
KEYWORDS: nitrification; ammonium; nitrate
REVIEW: N. Nobody; nnobody@wiz.uni-kassel.de;
SIMTYPE: event
CONSTANT:
  F[n]: probability of fertility
  P[n]: probability for survival
INPUT:
  POP_0[n]: initial distribution of number of individuals
  n: number of age classes; type= IM
  MN: mineral nitrogen content of soil;
  XC[3,4]: substrate carbon;
COMPONENTS:
  component[1]=soillayer_0;
  component[2-9]=soillayer_1;
PROCEDURE:
  step(A,B->C,D);
  C=A+B
  D=A-B✍A>B);A*B
FUNCTION:
  funci(x,y,z);
  funci=x+y^2-z
CONSTRAINTS: a<b
CONDITION: x==25
START: t==0
REF: L99;
AUTHOR= Schwinning S.;
AUTHOR= Parsons A.J.;
TITLE_ART= Analysis of the coexistence mechanisms for
grasses and legumes in grazing systems;
Specification section
QUANTITIES:
CONSTANT:
  NH_4: g*m-2; water; value=0.3
INPUT:
  XC0: g*m-2; water
STATE:
  XC: g*m-2; water
TIME:
  t: s; time
SPACE:
  z: km; space
```

Figure 24: ECOBAS_MIF examples (BENZ, J; HOCH, R., 1999)

Finally, the Domain section is considered to be an extra section that includes model classifications from different application areas. In order to contribute to efficient retrieval of similar models, ECOBAS_MIF uses international standards. For instance, the FAO-classification (DRIESSEN, P. J., DUDAL, R., 1991) for soil type was previously registered by the system, so that when specifying a model, the user can choose the most appropriate one from a classification list. There are classification lists also for soil texture, climate type, ecosystem, and biological taxonomy available.

Both REM and ECOBAS-MIF structures include descriptors that contribute to a rich scientific model description. However, these structures do not explicitly represent relationships between models and their implementations, which difficult their management. Again, a metamodel design would help on representing all relationships explicitly. Moreover, scientific data, workflows and model use are not considered in ECOBAS-MIF as useful associated descriptions.

3.3 Managing Scientific Workflows and Registering their Use

Workflow Management Systems (WfMS) define, manage and execute workflows through the execution of software whose order of execution is driven by a computer representation of workflow logic (WFMC, 1999). WfMS basic functionality includes workflow definition, workflow instantiation and execution. The workflow logic is the result of the workflow definition. It corresponds to the specification of all necessary information about the process to guide the workflow instantiation and execution. To instantiate a workflow the WfMS interprets the workflow definition and occasionally asks the user for complementary data/information. As the instantiation proceeds the WfMS starts the workflow execution, invoking software programs when necessary.

To address the scientific community though, WfMS basic functionality is not enough. Interoperability, integration, abstract workflow definition, dynamic definition, and workflow auditing are some of the required extra functions. Interoperability may be provided through the association with a middleware layer responsible for invoking remote applications and data, as well as the communication ability with other WfMS. When dealing with multi-domain and multi-platform applications, as it is the case within the scientific community, the WfMS must provide some metadata support for resolving heterogeneity conflicts and to allow the integration of such systems.

Another important feature with respect to scientific workflow management is the possibility of defining abstract workflows. A workflow can be defined in terms of task classes or their abstract description. Alternative task instances can be considered to be equivalent when they belong to the same task class or abstract description. The workflow instantiation and execution module can benefit from these layered service descriptions, as it can choose from a list of available candidate tasks provided by the task class directory. This late binding allows the instantiation module to use alternative instance tasks when some are not available. Moreover, scientific workflows are usually not fully specified before it starts being instantiated. Some definitions are left for the instantiation moment. Thus, WfMS need to support workflow dynamic definition, i.e., allow workflow definitions to be modified, as they are instantiated, demanding a tight evolution control.

WfMS workflow auditing facility involves tracing workflow instantiations. *In silico* experiments are directly associated to workflow instantiations. To register scientific workflows use means to document a scientific experiment. Specific code executions, with specific input and output data resources, and specific parameter values are all part of a scientific experiment. In the case of scientific workflows, WfMS should report all scientific resources involved in each workflow instance associated to a scientific experiment.

Most of the time, WfMS adopt a task-centric approach that is reflected by their architecture, in which a Database Management System (DBMS) is used to store task descriptions, and which includes all workflow functionality in modules that run on top of the DBMS. However, in scientific workflows data set descriptions are as important as the description of tasks that processes them because the quality of an input data set often impacts the quality of the output of the program that has processed it. Furthermore, the quality of data used and generated along an experiment influences the workflow instantiation associated to that experiment. Therefore, to address scientific workflows a WfMS should combine task and data-centric approaches.

Web services are already heading to workflow management, however, as a generic purpose technology it needs additional facilities to fully address scientific workflows management. On the other hand, some research projects were specifically conceived to address scientific workflows and scientific resources use. Although these initiatives do not address all the problems raised here, they represent relevant contributions.

In the next sub-sections (3.3.1, 3.3.2, 3.3.3 and 3.3.4) we revisit some of the projects discussed so far, with respect to their support for workflow management and registry. Le Select and ECOBAS-MIF do not address these issues, and are not discussed here. Other three initiatives that especially address scientific workflow management are briefly discussed: ZOO (section 3.3.5), WASA (section 3.3.6) and AGROMET (section 3.3.7).

3.3.1 WSA and OGSA

Web services architecture also provides the necessary mechanisms to define workflow processes through the composition of Web services. Such compositions are defined through XML-based languages expressing the data should be processed across a collection of Web services, just like traditional workflow specification languages. Therefore, through the use of Web services technologies, an e-scientist can define scientific workflows based on the specification of standard and reusable Web services.

A number of proposals for such Web services composition language came from the major industry players. However, recently, some of them came to an agreement (IBM, Microsoft and BEA) and released BPEL4WS – Business Process Execution Language for Web services (CURBERA, F., GOLAND, Y, *et al.*, 2002). BPEL4WS is a XML-based language for coordinating business process over the Web, which relies on Web services technology. Since BPEL4WS is the first joint industrial effort to define a specification for Web services composition, it is a strong candidate to become the standard language for specifying Web services compositions. BPEL4WS provides a language to formally specify a business processes and business interaction protocols. It extends the interaction model of WSDL to define a process that provides and consumes multiple Web services interfaces.

With respect to scientific workflow management, interoperability and integration are already provided by Web services. As a WSDL document may consist of abstract definitions of services, we may say then that BPEL4WS is able to describe abstract workflows. However, abstract workflow instantiation, workflow dynamic definition and auditing are up to the workflow processing engines.

Finally, as OGSA is based on Web services architecture, it can also benefit from the facilities provided by WSA described here.

3.3.2 MyGrid Project

MyGrid (WROE, C. *et al.*, 2003) recognizes that bioinformatics scientists need to tie code resources together into scientific workflows. According to MyGrid architecture (Figure 13) the workflow modules provide the basic WfMS functionality, and adds to it abstract workflow definition, instantiation and execution. So far, MyGrid project uses the workflow language WSFL (WSFL) for workflow definitions. MyGrid allows for workflow reuse, taking a user definition and comparing it to previously authored definitions. Some of these workflows may have fixed instance services (pre-set binding), others profit from the late binding feature (abstract workflow definition). Once the user has selected the appropriate workflow definition and data, MyGrid counts on existing WSFL processing engines to execute them. Finally, the workflow results are stored and available for future processing. As MyGrid plans to count on the Web services infrastructure, interoperability and integration of data and programs are provided.

In their work (WROE, C. *et al.*, 2003), the authors claim to track experiments and to document them so they can be re-executed and data provenance can be available (workflow auditing). MyGrid registers invoked instance service descriptions, which include information about the execution of a particular code resource, on a particular date, using particular parameter values. However, it is not clear how these executions are registered in the scientist personal repository, that is, if there is a metamodel supporting the registry. Moreover, it is also unclear if the registered resources can be exchanged with other scientists.

3.3.3 GriPhyN Project

The GriPhyN Project has a special interest in registering scientific resources use. The Chimera System Data Schema (FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003) includes concepts that are specially focused on data lineage: invocation and derivation concepts. These concepts are responsible for keeping data lineage information on a metadata repository. Through them it is possible to document which dataset matched which program input. Also, it is possible to keep track of which program has generated a specific dataset. However, it does not provide workflow management facilities.

3.3.4 ESSW and ESP2Net Projects

Although no workflow management facilities are provided, ESSW was specially conceived to collect metadata about scientific experiments involving program executions. The LN database shown in Figure 17 captures metadata about science objects such as executable models (program codes) and data types. These metadata is then used to register program executions. There is no support for workflow definitions, but workflow instances are implicitly registered as the execution of a sequence of programs.

ESSW provides visualization tools to map these workflow instances in terms of each science object and their connections, as a **directed acyclic graph (DAG)** showing the linked steps, as example in Figure 25 shows. Each step is described based on previously described science objects, e.g. models (rectangles), and their inputs and outputs (circles).

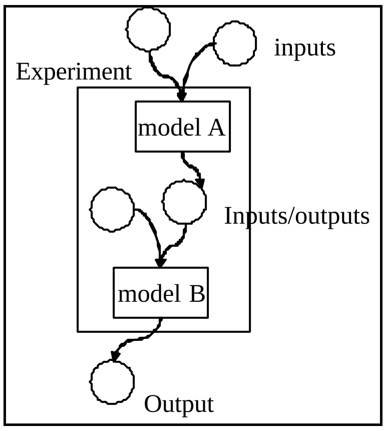


Figure 25: ESSW Experiment

Similar to ESSW, the ESP2Net project focus is on interchanging scientific datasets, by publishing the experiments that generated them. However, it also does not provide workflow management facilities.

3.3.5 ZOO System

The ZOO system (IOANNIDIS, Y.; LIVNY, M.; GUPTA, S.; PONNEKANTI, N., 1996) is a desktop experiment management environment. It supports domain-specific teams of scientists, although the development has taken a generic approach. At the core of ZOO system, the workflow is viewed as a Web of data objects interconnected with active links that carry process description. In Ailamaki et al. (AILAMAKI, A.; IOANNIDIS, Y.; LIVNY, M., 1998), the DBMS incorporates the WfMS functionality, representing workflows as schemas. Workflow dynamic definition and auditing are more easily provided. Although ZOO´s centralized approach provides benefits, such as a unique access language and point of control, it does not address distributed and heterogeneous environments, which is the case of scientific applications. Also, abstract workflow definitions are not supported.

3.3.6 WASA

Another interesting work that considers scientific WfMSs (WESKE, M.; VOSSEN, G.; MEDEIROS, C., 1996) describes the WASA architecture, whose goal is to provide a supportive environment for data-intensive scientific applications. WASA's main contributions are the support for dynamic execution of tasks (workflow dynamic definition), by combining active and temporal database facilities, and the support for experiment re-usability (auditing) and reproducibility, by means of the documentation and versioning facilities. Although WASA can be seen as a generic architecture for scientific workflows, it was not developed for distributed and heterogeneous scientific programs, lacking interoperability facilities. Also, it is not clear whether WASA is able to deal with abstract workflows.

3.3.7 AGROMET

AGROMET system (PINTO *et al.,* 2002) supports scientific work in a cooperative way, involving document, knowledge and workflow management. AGROMET provides a data integration middleware to support interoperability between autonomous distributed and heterogeneous data repositories. The workflow management module supports the definition of abstract workflows and provides experiment reuse, stored in an experiment base (workflow auditing). In addition, AGROMET includes a cooperative work support module on top of all the other modules, to help on collaborative analysis in scientific projects. However, scientific applications demand support for distributed and heterogeneous programs, which is not addressed by AGROMET.

3.4 Discussion

Scientific resources management systems presented here aim to address scientific applications by handling distribution and heterogeneity of scientific resources, describing these resources, managing scientific workflows and registering their use. Most of these technological and project approaches bring relevant contributions to scientific applications. These efforts adopt different strategies to deal with scientific resources management. In some cases, the emphasis is on distribution and heterogeneity, without taking into account higher level description facilities such as models and experiments. In other cases, the focus is on workflow, without providing program distribution and heterogeneity. Also, a lack of workflow instances and experiments registry facilities are frequently found, making it hard to rerun previous experiments or obtain data provenance. Figure 26 presents a table that summarizes all the systems described in the last sections, indicating the facilities they provide (✓) or not (✗), or yet whether it is not reported (?), according to those three issues.

With respect to the handling distribution and heterogeneity issue, we first define which Client/Server technology is used. Grid and Web services (WS), as well as JDBC and CORBA, are some of the technologies used. Some projects/initiatives use a proprietary technology solution or none at all. Then we describe each system facilities with respect to data and program distribution handling, and data and program heterogeneity handling. We also describe their ability for remote execution and execution control. Remote execution means the code may be executed on another host machine, and that the system will be able to handle data transfer. Execution control means to be able to monitor a service execution, asking about its status while it runs. A high level of control would enable to cancel an execution.

The description facilities include the ability of each project or system to represent scientific resources. There two main approaches adopted by these projects to address these facilities: the metamodel-based (MM) approach and the ontology-based (Ont) approach. As a first desired facility, we have identified the need to represent scientific models, and distinguish them from their implementations (programs) and available compilations (codes). So, codes should be related to a program description, which should relate to its input/output, classified under data types. Analogously, scientific data sets should also be described independently of programs, and related to data types. Another desirable description facility is to define data replicas, monitoring the use of alternative equivalent datasets. Finally, description extensibility means to be able to provide description extensions, according to some specific-domain users.

To provide workflow facilities means to be able to manage scientific workflows, which includes: workflow definition, abstract workflow definition, workflow execution, dynamic workflow definition, and workflow partial execution. To provide registry facilities means to be able to provide data provenance, i.e., when a code execution has started, which data input/output was used, etc. A code execution registry means to register every single code execution individually, while the workflow instance registry means to have it registered as set of linked executions. Finally, an experiment registry means to have a set of workflow instance registries as belonging to a specific experiment.

	Le Select	WSA	OGSA	MyGrid	GryPhiN	ESSW	ESP2Net	ECOBAS-MIF	ZOO	WASA	AGROMET
Distribution and Heterogeneity facilities											
C/S Technology	JDBC/ CORBA	WS	WS Grid	WS Grid	Grid	JDBC	CORBA JRMI	--	--	Prop	Prop
Data Distribution	✍	✍	✍	✍	✍	×	✍	×	×	✍	✍
Data Heterogeneity	✍	✍	✍	✍	✍	×	×	×	×	✍	✍
Program Distribution	✍	✍	✍	✍	✍	×	✍	×	×	×	×
Program Heterogeneity	✍	✍	✍	✍	✍	×	×	×	×	×	×
Prog. remote execution	✍	✍	✍	✍	✍	×	✍	×	×	×	×
Execution control	✍	×	✍	✍	✍	×	?	×	Some	Some	?
Description facilities											
Approach	--	--	--	Ont	MM	MM	MM	--	MM	MM	MM
Model description	×	×	×	×	×	×	×	✍	×	×	×
Program description	×	✍	✍	✍	✍	×	×	✍	✍	?	?
Program IO description	×	✍	✍	✍	×	×	×	✍	✍	?	?
Data Type description	×	✍	✍	✍	✍	×	×	✍	✍	?	?
Code description	✍	✍	✍	✍	✍	✍	✍	✍	✍	✍	✍
Data description	✍	✍	✍	✍	✍	✍	✍	×	✍	✍	✍
Data replica description	×	×	×	×	✍	×	×	×	×	×	×
Description Extensibility	×	✍	✍	✍	?	✍	×	×	×	×	×
Workflow and Registry facilities											
Wf definition	×	✍	✍	✍	✍	Scripts	×	×	✍	✍	✍
Abstract wf definition	×	✍	✍	✍	✍	×	×	×	?	×	✍
Wf dynamic definition	×	×	×	×	×	×	×	×	✍	✍	×
Wf execution	×	×	×	✍	×	Run scripts	×	×	✍	✍	✍
Wf partial execution	×	×	×	×	×	×	×	×	?	✍	?
Code exec. Registry	×	×	×	✍	✍	✍	✍	×	✍	✍	✍
Wf instance registry	×	×	×	×	×	✍	✍	×	✍	✍	✍
Experiment registry	×	×	×	×	×	×	✍	×	×	×	×

Figure 26: Comparative Analysis of Related Work

With respect to the distribution and heterogeneity technology, Le Select and Web services architecture have taken distinct paths, but they ended up sharing the same objective: to provide a cross-platform approach to component-based development. Even though Le Select came first, Le Select is not a technological middleware such as Web services. Le Select adds software layers for database management. It also proposes a Le Select closed community, similarly to the Grid community, whereas Web services is a Web open architecture proposed by W3C (WS), and therefore more easily adopted by the market, and particularly by the scientific community.

Furthermore, Web services architecture goes further than Le Select, providing a service catalog provider (discussed in section 3.2), which allows service requesters to find the service they need. Grid computing, currently merging to Web services through OGSA, is also sharing these benefits. Some of the main Component-based Development Environments in the market, such as BEA WebLogic and IBM WebSphere for Sun J2EE platform, and the Microsoft .NET platform, are already considering Web Services. The idea is to open their environments to provide, alternatively, Web services compliant components, i.e., a component that could also interoperate with components from any other client/platform. However, none of them alone can provide all the semantic description that scientific resources need.

While WSA, OGSA, MyGrid and GryPhiN are close to what scientific resources need in terms of description, they do not describe scientific models nor do they cover experiment registry completely. ECOBAS-MIF fails completely in terms of experiment registry, while it is the only one that recognizes scientific models as important resources to describe. On the other hand, projects like ESSW and ESP2Net fail in terms of description, but concentrate on experiment registry. Finally, with respect to which description facilities ZOO, WASA and AGROMET provide, it is important to mention that, as their corresponding metamodels were not explicitly published (to the best of our knowledge), most of the table cells in this section carry some degree of uncertainty.

Other interesting scientific resources description proposals can be found in the literature (HOUSTIS, C.; LALIS, S., 2001) (CRITCHLOW, T.; MUSICK, R.; SLEZAK, T., 2001). However, they also fail on presenting a complete metamodel or functionality to address scientific resources facility.

There are basically two approaches for describing scientific resources: metamodel and ontology-based. We believe that a metamodel-based approach is best suited to describe scientific resources, however, none of the proposals found so far have completely addressed the requirements for this kind of application. The metamodel-based approach is particularly good because its focus is on the development of management applications. In this approach, several conceptual levels are captured in a metamodel, and resources are described according to this metamodel. In the ontology approach, concepts and resources are equally represented in specific-domain ontologies. In this approach, there is not a clear distinction between the resource and description levels. In contrast with ontology flexibility, the metamodel approach counts on a pre-defined structure, which guarantees that each resource is described according to it. This is evidenced in MyGrid project where several "independent" ontologies, one for each "context", are built and later a different ontology has to be built to allow navigation from one context to another. For instance, the extended DAML-S ontology is used to allow navigation from services to bioinformatics ontologies. This navigation is an inherent concept of the metamodel approach.

Metamodel and ontology approaches are not conflicting; in fact they are complementary (SPYNS, P, MEERSMAN, R., JARRAR, M., 2002). We believe that it is possible to combine both approaches, to get the best each one can offer. While the metamodel provides the basis for navigation through scientific resources, ontologies helps on the search and classification of each resource described by the metamodel. However, it is worth to mention that building ontologies requires a hard and specialized effort that should be done for each different scientific application domain. Moreover, there is not an established methodology to support ontology building, which compromises the quality of the existing ontologies. Therefore, we believe that scientific management will benefit from the metamodel approach, in a short term. These initiatives could later on be improved as the ontology expertise and supporting methodologies evolve.

The next section presents our metamodel-based solution to address scientific resources management. In our proposal we have concentrated on the metamodel development, and combined it with the most recent middleware technology to provide the required functionality.

4. Specification of a Scientific Resources Management Infrastructure

A scientific resources management infrastructure should be provided to address scientific applications. In section 3 we specified three main requirements that a scientific resources management architecture should provide: (i) handling distribution and heterogeneity, (ii) describing scientific resources, (iii) managing scientific workflows and registering its use. Among these three, we consider resource description requirement the fundamental issue that impacts all the others. Despite many recent initiatives in attending these requirements, so far we have not found any architecture that offers support for all three with a strong emphasis in metadata issues. Therefore, we present here an architecture with focus on a metamodel, to manage scientific resources available throughout the Web, where scientists are able to publish models for direct real case usage.

We have proposed the Scientific Resources Management (SRM) architecture (CAVALCANTI, M. *et al.*, 2002 b) where some of these mechanisms are provided. Here, we extend SRM to encompass scientific workflows and experiments support.

SRM is based on a metamodel approach. The Scientific Publishing Metamodel (SPM) (CAVALCANTI, M. *et al.*, 2002 a) is a fundamental feature to provide support to the SRM architecture. In section 4.1 we present the SRM architecture. Section 4.2 presents the SPM metamodel that offers the foundation for SRM architecture, describing each of the concepts it represents.

4.1 SRM Architecture

The Scientific Resources Management architecture addresses most of the problems raised in section 2.4 and 3 by the combination of five modules: Navigation, Publication, Experimentation, Resource Operation, and Resource Description modules. Referring to the three main requirements listed previously, the Resource Operation module addresses item (i), while the Publication and Resource Description modules address item (ii). Finally, the Experimentation and Publication modules address item (iii).

Figure 27 shows SRM has two layers: one for enabling Web access services and the other for scientific resources management. There are two main modules to manage scientific resources: the Resource Operation module and the Resource Description module. The Resource Operation module (RO) deals with data and programs. The Resource Description module (RD) is a metadata repository manager, dealing with data and program descriptions, and also with model, experiment and workflow descriptions.

The Web Access layer (WAL) is composed of three other modules: Publication, Experimentation and Navigation. The Navigation module allows scientists to browse scientific resources and their correspondent descriptions. The Experimentation module allows the user to perform *in silico* experiments. Within an experiment, the user is able to choose, instantiate and execute workflows specifications, which may be composed of programs or models. The Experimentation module interacts with the user, helping him/her on the workflow instantiation process. Then, if the workflow is completely defined, the Experimentation module interacts with the Resource Operation module by issuing remote execution messages with the specified input data. After this, the Experimentation module keeps track of the ongoing experiments, by publishing each essay. In summary, the Experimentation module guides the user on the correct use of the available models, providing an on-the-fly interface for executing them. Finally, the Publication module is responsible for publishing scientific resources. When a publisher enters some resource descriptions, the module checks these inputs by interacting with both the Resource Operation and Resource Description modules. Once validated, the Resource Description module stores these inputs.

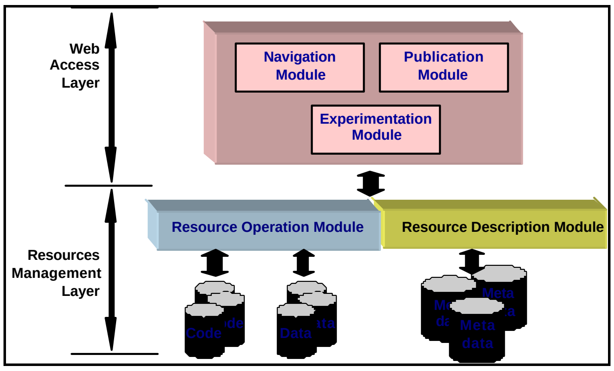


Figure 27: SRM Architecture

SRM is a distributed architecture (Figure 28). The Web Access layer (WAL: Navigation, Publication and Execution) corresponds to the client side of SRM, and provides access to the other two modules, the Resource Description (RD) and Resource

Operation (RO) modules, which act as servers. An SRM client is able to connect to one or more Resource Description and Resource Operation modules. Each RD server may store descriptions about resources served in more than one RO servers. The idea is to organize RD servers of related resources. In this scenario, the SRM client connects itself to one or more RD servers to access resources descriptions. According to the user choices, the SRM client then connects to one or more RO servers.

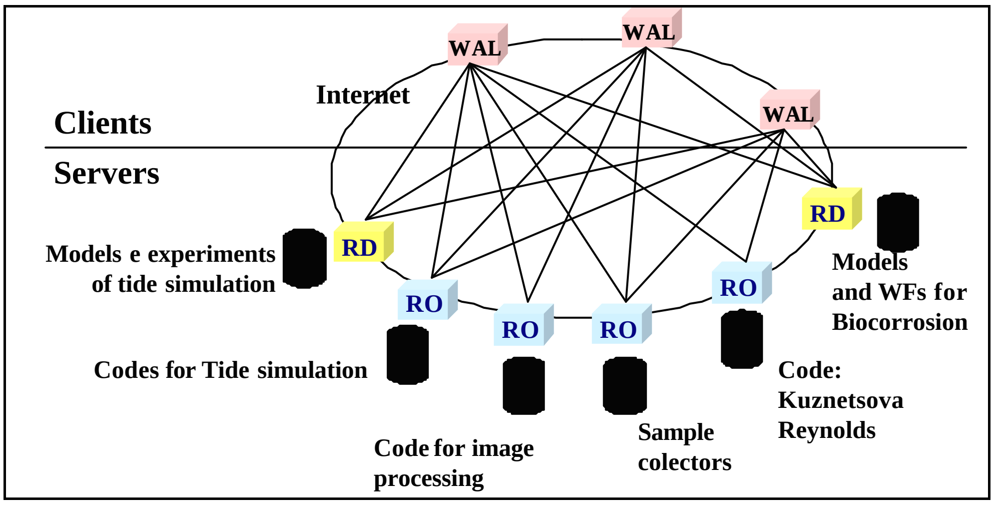

WAL
WAL
WAL
Internet
Clients
Servers
RD
RD
Models e experiments of tide simulation
Models and WFs for Biocorrosion
RO
RO
RO
RO
Codes for Tide simulation
Code: Kuznetsova Reynolds
Sample colectors
Code for image processing

Figure 28: SRM Architecture, a distributed View.

4.1.1 Resource Operation Module

In SRM architecture there are two main roles: the publisher and the user. The user basically navigates through described resources, trying to find some useful resource, and then actually accesses it. On the other hand, the publisher is basically a resource provider. Therefore, the publisher role is in charge of providing a Resource Operation (RO) module, which actually accesses data and executes code. In general, each RO module may be serving a set of data/code resources, grouped according to their location or platform. Each RO module works like a wrapper that allows the code or data to have a uniform interface in accordance with the distributed environment standard.

The Experimentation module receives requests for data retrieval and for code execution. As shown in Figure 29, it provides resources requestors that establish connections with RO modules. The idea of the SRM architecture is to work on top of the Web infra-structure. Therefore, the RO module should embed a Web server, which interacts with the Execution module through Web protocols. At the other end of the RO server is the legacy code and its invocation is specific. However, the RO module aims to be a generic way of enabling the invocation of any code. To address this goal, the RO module provides an extra layer of interaction between the code requestor and the legacy code itself. Within this extra layer, a generic invocation protocol is used to invoke the legacy code. Besides the Web server, other two facilities are necessary to provide this generic invocation: a message router and a code wrapper. The message router is used to process the generic invocation protocol and to invoke the code wrapper. The code wrapper is what has to be built to deal with the legacy code specificity, i.e., it knows how to invoke the legacy code and how to deal with its results, sending them back to the message router using the same generic protocol.

Data access is similar to code invocation. Each data set is stored according to a data model, which either may be associated to a specific query language, or provide a specific query interface application to manipulate its content. In this case, a data wrapper is also necessary to deal with data sources heterogeneity. The SRM architecture assumes a standard data model as the generic interface between the data requestor and the data repository. The message router is used to process the generic data invocation protocol and to invoke the data wrapper. The data wrapper knows how to access the data repository, and how to transform them into the standard data model expected by the caller.

Despite the different data models available, data sources are not as heterogeneous as legacy code. There are many data sources that use the same data model (e.g., the relational model). In this case, standard data wrappers can be built to help the publisher in providing an RO server for a data resource. Within the scientific scenario though, scientists commonly use structured text files. In this case, specific data wrappers are needed.

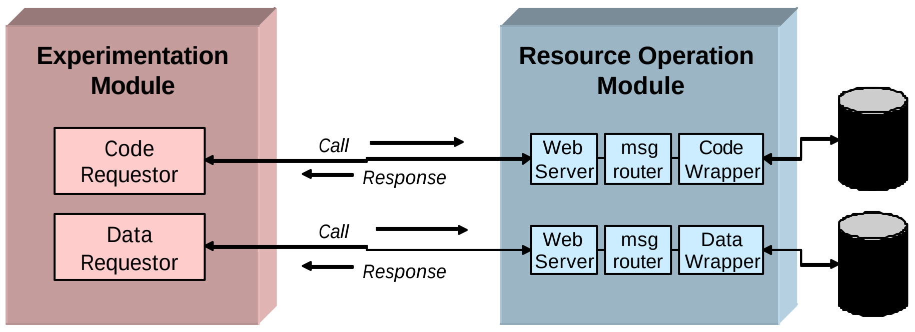


Figure 29: Resource Operation module

4.1.2 Resource Description Module

To allow SRM users to access data or code resources provided by RO modules, each resource must be described in the Resource Description (RD) module. The publisher may either provide an RD module or use an existing one. It is natural to have a group of related distributed data/code resources described in one RD server.

The RD module manages metadata of scientific resources. The Scientific Publication Model (SPM) is the metamodel (schema) behind the Resource Description module, and is described in more detail in Section 4.2. Each resource description is stored in accordance with the SPM. Considering some of the requirements raised before, XML seems to be the most adequate language to express scientific resources descriptions. Therefore, the SPM is expressed as an XML Schema. The idea of having a metadata repository manager came from the need to store semi-structured descriptions expressed in XML.

The RD module (Figure 30) includes a DBMS server. XML-enabled DBMS and native XML DBMS are both alternatives for XML storage. In the case of XML-enabled database systems, to guarantee that XML documents will be properly stored in that database, it is necessary to use or extend existing XML APIs. Therefore, the RD module embeds not only a DBMS server, but also a client application that includes an XML API.

In the RD module, the DBMS server maintains three main collections: XML, XSD, and Xix. The XML database stores all XML documents with metadata about any scientific resources area, e.g. Biology, Oceanography, etc. The XSD database stores XML Schemas used for XML documents validation. In this architecture, XSD contains the SPM XML Schema, which specifies how scientific resources such as data category, transformation, data, code and experiment should be described. Finally, the Xix database stores an index over the XML documents to provide direct access to these documents, facilitating keyword searches. The Indexer facility is responsible for building the Xix database by indexing XML documents stored in the DBMS.

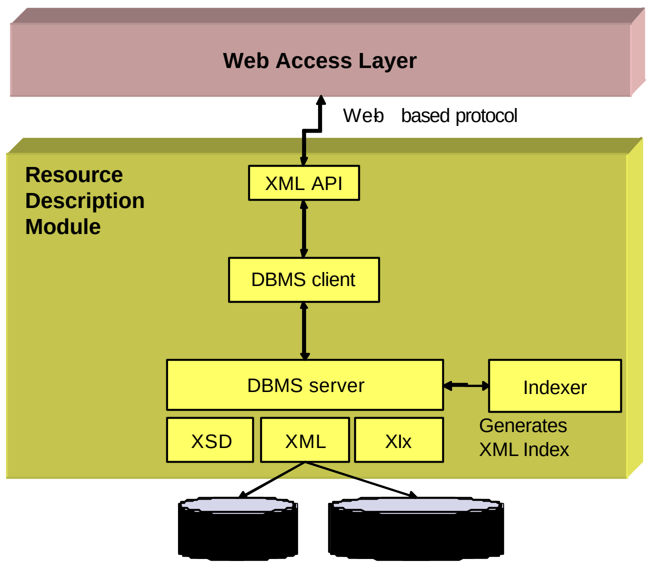


Figure 30: Resource Description module

Each module of the Web Access layer (WAL) interacts with the RD module through Web-based protocols. The Publication module (Section 4.1.3) feeds the RD module with XML documents, while the Navigation module (section 4.1.4) interacts with it by issuing queries and retrieving XML documents for navigation. Finally, the Experimentation module (Section 4.1.5) issues queries to the RD module for validating code executions, and then feeds it with experiment descriptions. The XML API is responsible for retrieving and returning XML documents that satisfy the issued queries.

4.1.3 Publication Module

The Publication module is responsible for scientific resource description capture. According to the SPM metamodel, each scientific resource has specific metadata that describes it. Therefore, the Publication module (Figure 31) provides a different entry form for each of them, through the Resource Publication Interface facility. These forms are built based on the SPM, which is expressed as an XML Schema. In fact, considering the evolutionary nature of metamodels, especially in the scientific scenario, we have designed the Publication module to be adaptable to different (versions of) metamodels. As we chose to have the metamodel stored in the metadata repository, it may be used not only to validate XML document instances but also for building the user interface.

The publication process involves two main scenarios. In the first scenario, the publisher is usually a scientist that has been publishing theoretic model resources within a scientific community. The publisher starts publishing *models* and their inputs and outputs, by filling up the correspondent forms. In this scenario, the publication is a top-down process, i.e., it starts with the more abstract description. In the second scenario, the publisher is either a *data* provider or a *code* provider. In this scenario, the process is bottom-up, and the publisher describes each code or data by publishing *programs* and the correspondent inputs and outputs. The association between models and programs happens in the bottom-up scenario, after having the program and model already published.

According to the bottom-up publication scenario, the Publication module first requires the resource operation address. Based on metadata about the resources available at the RO module, the Resource Selector provides a list of the served code/data resources, where the publisher selects and publishes it in the RD module. However, a code/data publication means to describe them as valuable scientific resources, i.e., program and model publications are also required. These descriptions are captured by the SPM Schema, therefore, the Schema Processor asks the RD module for the correspondent XML schema subset and prepares it for the Interface Builder. Based on the schema key reference element definitions, the Schema Processor transforms the abstract relationship between resources into an enumeration type with references to identifiers of document instances. After prepared, the schema is delivered to the Interface Builder which transforms it into an HTML page, where a publication form is ready to be filled up. Finally, when the user submits a form to the Form Handler facility, the submitted data are translated into an XML document and sent to the XML Validator facility to be validated against the Schema.

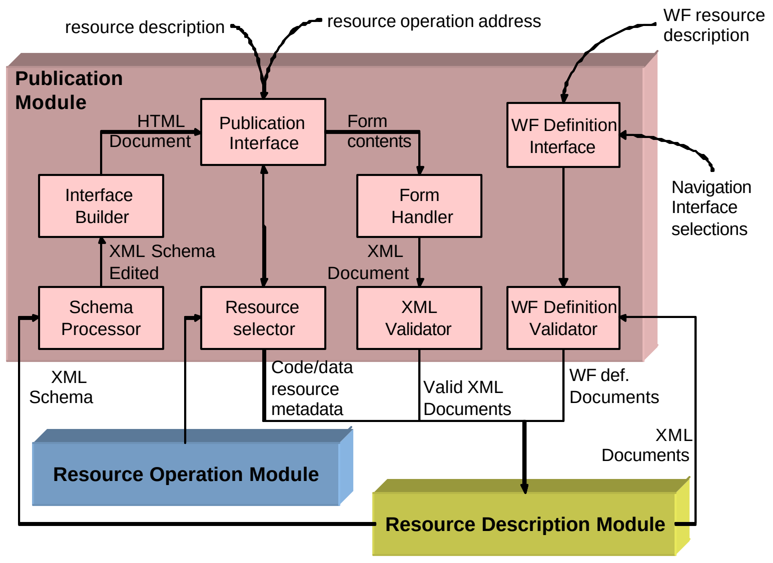


Figure 31: Publication module

When guiding the user through the top-down scenario, the Publication module jumps directly to the interaction with the Schema Processor, which asks the RD module for the correspondent XML schema subset and prepares it for the Interface Builder.

The workflow publication starts at the navigation process. While navigating, the user selects which programs/models are needed. The Workflow Definition Interface facility captures the navigation selections and interacts with the user for the workflow definition. Then, to guarantee that the workflow definition combines compatible programs/models, the WF Definition Validator facility requests for program/model descriptions stored in the RD module. Finally, after validated, the workflow is published in the RD module.

4.1.4 Navigation Module

A quick glance at the most usual queries suggests that the scientific user searches for different data characteristics, such as substance names (e.g. Calcium), quantities (e.g. concentration) and units (e.g. mg/l). The Resource Description module usually answers these queries. However, due to the diversity of scientific users, there is not a pre-defined way to present such queries. Therefore, the need for a keyword-based search facility is clearly identified. Also, a guided navigation is required; based or not on the results of a keyword search request. Dynamically configured interface pages should guide the user through selected available resources.

The Navigation module is responsible for handling user queries and navigation over/through scientific resources. Therefore, the Navigation module includes facilities for querying XML documents stored by the Resource Description module and for handling query results to be presented to the user (Figure 32). Depending on the request of the user, the Query Interface facility may submit it either to the Keyword Search Handler or to the Query Handler.

The Resource Description module takes advantage from the built-in indexes (Xix) to retrieve a set of possible XML document references, which are processed and built by the XML Processor and Interface Builder facilities, respectively, and then sent back to the user.

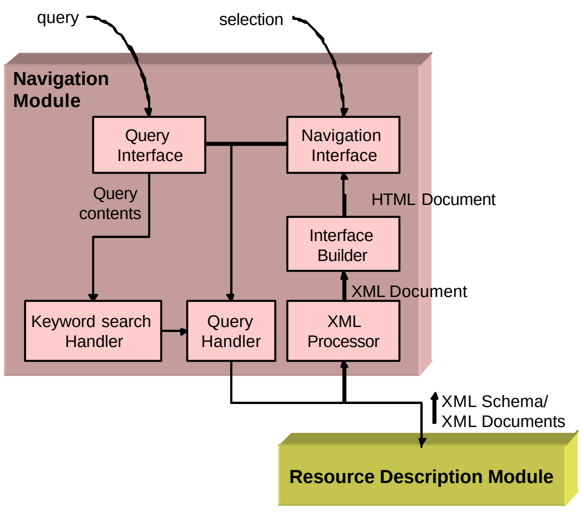


Figure 32: Navigation module

The Navigation module also provides some pages through which the user can navigate and ask for more specific information on available resources. These user requests are passed to the Query Handler facility and then submitted to the RD module. The result of such requests may either be a set of references or a single XML document. Both are returned to the XML processor, which adds links to other documents based on the XML Schema relationship information.

4.1.5 Experimentation Module

An experiment begins with its description, which is provided using the Publication module. After describing the purpose, hypothesis and workflows associated to an experiment, the user may then start it. The Experimentation module is responsible for the experiment management, and it counts with the facilities shown in Figure 33. The user starts an experiment through the Experiment Initiation Interface facility by choosing which workflow, associated to the experiment, is going to be instantiated.

The next step is the definition of a concrete workflow. The workflow published so far is an abstract workflow, and a corresponding concrete workflow is needed to be available for execution. The Concrete WF Definition Interface facility (Figure 33) supports the concrete workflow definition, which involves choosing data and code resources to be used according to the abstract definition. If the abstract definition is at the model level, then there must be two choice levels. The first level will choose which program is best for a model, and the second level will choose which code resource is best for that program. Data resources are chosen according to the input data categories and constraints associated to the programs chosen, i.e., the user navigates through a list of data resources that are compatible to the input data categories of the workflow programs. With respect to the choice for code resources, the code/data definition facility may either provide a list for user navigation and choice, or count on the definition of some selection criteria. Differently from data resources, code resources are all equivalent in terms of functionality, as they are compilations of the same program. Therefore, instead of a direct choice, specific criteria, such as cost or availability, could be defined for a further automatic choice of which would be the best code resource. Finally, parameter values are also defined at this point, either by user direct input, or by choosing a data resource.

When ready, the concrete workflow definition is passed to the Workflow Engine facility. First, the facility publishes this definition in the RD module as a registry of the on going essay, and then starts processing it. If code resource criteria are defined, the engine should be able to "resolve" that by, for instance, computing costs or finding out the code availability. For each data and code resource specified in the workflow definition, a corresponding code/data requestor is called, which interacts directly with the corresponding RO module. The data requestor should be able to select data according to program input constraints. Code requestors should be able to send the selected data to the corresponding RO module. After finishing each code execution, data results are temporarily published as data resources, and when it is the case, passed to the next step in the workflow execution, i.e., another code requestor. Each finished code execution is registered in the RD module, composing an essay publication.

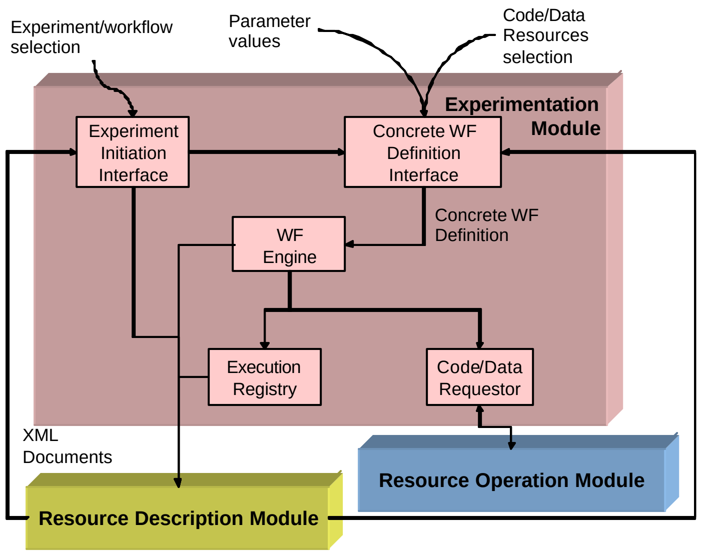

Experiment/workflow selection
Parameter values
Code/Data Resources selection
Experimentation Module
Experiment Initiation Interface
Concrete WF Definition Interface
Concrete WF Definition
WF Engine
Execution Registry
Code/Data Requestor
XML Documents
Resource Operation Module
Resource Description Module

Figure 33: Experimentation module

4.2 Scientific Publication Metamodel

The Scientific Publication Metamodel (SPM) addresses most of the description facilities presented in section 3.4. As SRM architecture follows a metamodel-based approach we have concentrated on the development of SPM to support this architecture. In a metamodel-based approach several concepts are captured in a metamodel, and resources are described according to this metamodel.

In this section, we define a set of concepts that form the SPM. Considering that scientific users and publishers basically deal with *programs* and *data*, we start with a generic approach, where we define two concepts (Figure 34) that can express not only what the user needs, but also what the publisher provides. *Data category* is the first concept that may be used by the publisher to *describe* his data and that may be used by the user to *define* the type of data he needs. Then, there is *Transformation category* that may be used by the publisher to *describe* what the functionality of his program is, and may be used by the user to *define* which kind of scientific solution he needs. Finally, the publisher may publish data as a data category and may publish program as a transformation category. Thus, users can improve their access, understanding and reuse of published resources, through these categories. In order to enhance readability, let us refer to Transformation category simply as *transformation*.

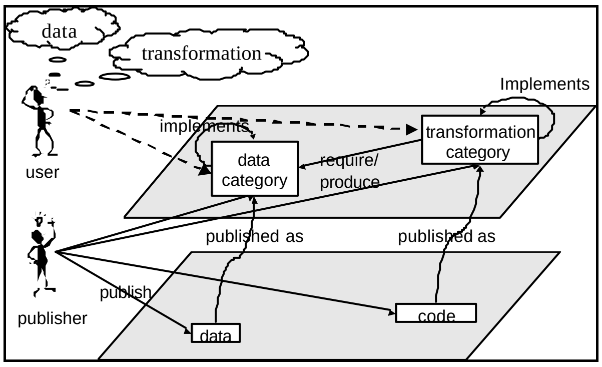


Figure 34: SPM Generic Concepts

The explicit definition of high level concepts like data category and transformation category is one of the main advantages of a metamodel-based approach. These concepts provide a common view of different but similar objects that are manipulated by different user roles, facilitating the interaction among them.

The following subsections present the SPM model in more details. Each SPM concept is expressed as a UML (UML REVISION TASK FORCE, 1999) class. SPM classes are related to each other within a class diagram. To facilitate SPM explanation, the complete SPM diagram was broken into subsets of related concepts. To see the complete diagram, please refer to the Appendix (section 9.1).

4.2.1 SPM concepts

To describe scientific data and programs, first it is necessary to "identify" them as a **resource**, which has a Web address and is described by a certain "type". In particular, as shown in Figure 35, a **data resource** is described by a **data category (ProgramDC)**, while a **code resource** is described by exactly one **program interface** (or simply, program)**.** A code resource represents one **code** that executes in a specific operational system and hardware, while a data resource represents one **data** set that was generated by some kind of mechanism, such as a satellite or a sensor, or even a code execution. To allow for data traceability, data provenance should be explicitly captured, determining the specific satellite or sensor identification, as well as the code execution that actually generated them.

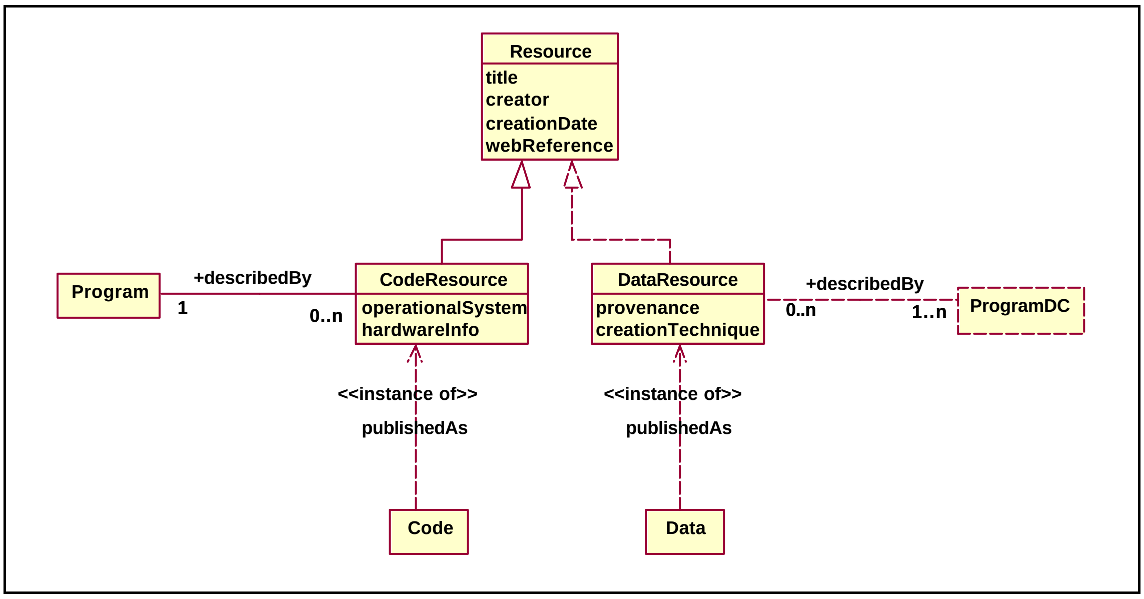


Figure 35: SPM operational resources related concepts

Usually, a scientific **program** is the implementation of a theoretic **model**. Both model and program concepts have many characteristics in common, although they belong to different usage levels. The program is actually executable, while the model is descriptive. To take advantage of such similarity, both model and program can be viewed as a **transformation**, as shown in Figure 36**.** A **transformation** is a description of a data transformation process that produces some output data and requires input data. Therefore, a transformation should be associated to at least one input and one output, and each one of such **I/O data** refers to a **data category**. An I/O data may be specialized as **input** or **output data.** Finally, a transformation is associated to a set of **parameters** and to a set of operational **constraints**, which express conditions on I/O data attributes and on transformation parameters.

Through the transformation abstraction it is possible to represent associations to I/O data, Parameters and Constraints for both concepts (models and programs). On the other hand, the differentiation between these concepts is important because it allows the representation of another important concept, which is the **implements** relationship. This representation allows the user navigation through models and programs, searching for similar programs that are based on the same model. In addition, models and programs are characterized differently. Models are described by a set of attributes that capture information that are typical of its level of abstraction, such as area, scope, purpose and hypothesis, while programs are described by implementation related attributes, such as the programming language and the program version. Also, programs are usually associated to more parameters than the model it is based on. These additional parameters are related to implementation issues such as performance and precision. Finally, the program is used to describe code resources, which reside in different hosts. When a user publishes a code as a code resource, he/she establishes an association of the code resource to a specific transformation, i.e., a program interface. This association will help the user to access, understand and use such code.

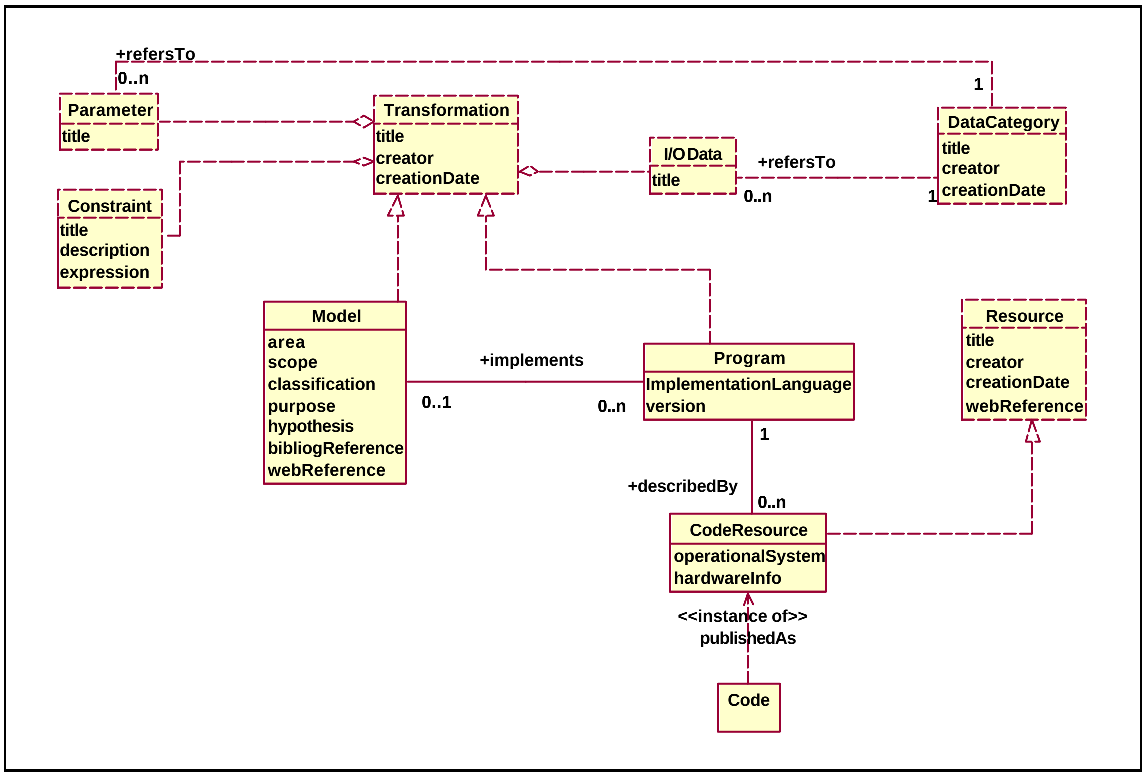


Figure 36: SPM transformation related concepts

Furthermore, SPM focus is on the description of the program and the theory behind it, the **model**. However, the focus is not to represent the model itself, such as a formula or an algorithm, but to describe it with adequate semantic to facilitate the decision of its adequacy to the problem in hands. Therefore, SPM represents relationships between models, such as **derivation** and **calibration**, shown in Figure 37, which allows for the tracing of a model lineage. The study of a family of related models facilitates the user on the evaluation of which one is the most adequate.

Scientists derive new models based on studies over some existing model. A study may consider, for instance, the influence of the temperature in the Kuznetsova model (KUZNETSOVA, V. A., 1960). Then, the new model includes some extra data input to accommodate temperature values.

A model calibrates another model when it is specifically conceived aiming at a particular situation, such as a specific geographic area. Usually, to calibrate a model means to have values assigned to some of its parameters. The result is still a model, but calibrated to a specific situation. Both models, generic and specific, may be described and also implemented. Let us consider the "straight line" example. The straight line equation takes the form $ax + by + c = 0$. The equation $5x - y + 2 = 0$ represents a specific straight line. In this example, the straight line is calibrated when parameters *a*, *b* and *c* assume values 5, -1 and 2, respectively.

The **ModelParmMatch** concept represents the relationship between the calibration and the calibrated model parameters. A simple type value may be assigned to each model parameter. Usually these values correspond to constant values, and can be represented by the ModelParmMatch attribute. As models are represented here as a descriptive concept, the idea is to document its calibration process and not to automate it.

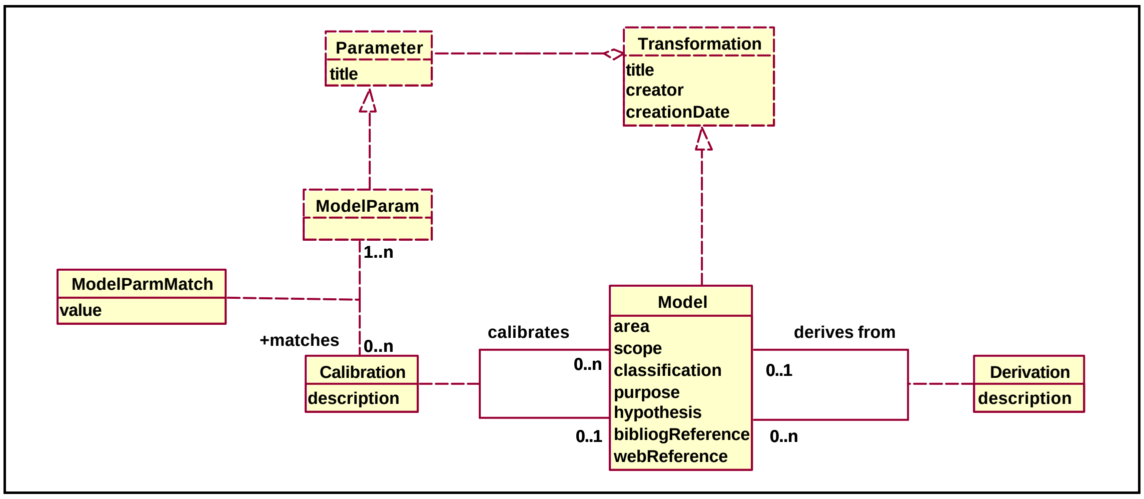


Figure 37: SPM model derivation and calibration concepts

Another important model descriptive attribute is its classification. There are many classification possibilities, described in section 2.3.1. When describing a scientific model it is not wise to choose one of them. On the other hand, a possible unification of all these classifications would be out of the scope of this work. Therefore, it seems more appropriate to allow the user to choose as many classifications as necessary to describe a model instance.

Figure 38 presents the data category and its related concepts. A **data category** describes scientific data that have some common characteristics. A set of **attributes** is used to describe each property of a data category. Some of these attributes are mandatory others are optional. A data category can be associated to a model, in this case it is called a model data category (**modelDC**); or it can be associated to a program, when it is called a program data category (**programDC**). The difference between these transformations becomes clear when we describe attributes of each data category associated to them. When describing data categories associated to the model, it is necessary to describe the quantity and/or classification of each attribute, while describing a program data category, each of its attributes needs a basic data type (e.g., integer), a unit (e.g., mg/l) and a format (e.g. GIF) specification. When publishing data as a data resource, the user associates it to a program data category. As real data is already committed to data units and formats, it does not make sense to describe it using directly a model data category.

The advantage of using model data categories at such abstract level of description facilitates the discovery of equivalent models. It is possible to find different purpose models, which use the same scalar quantities.

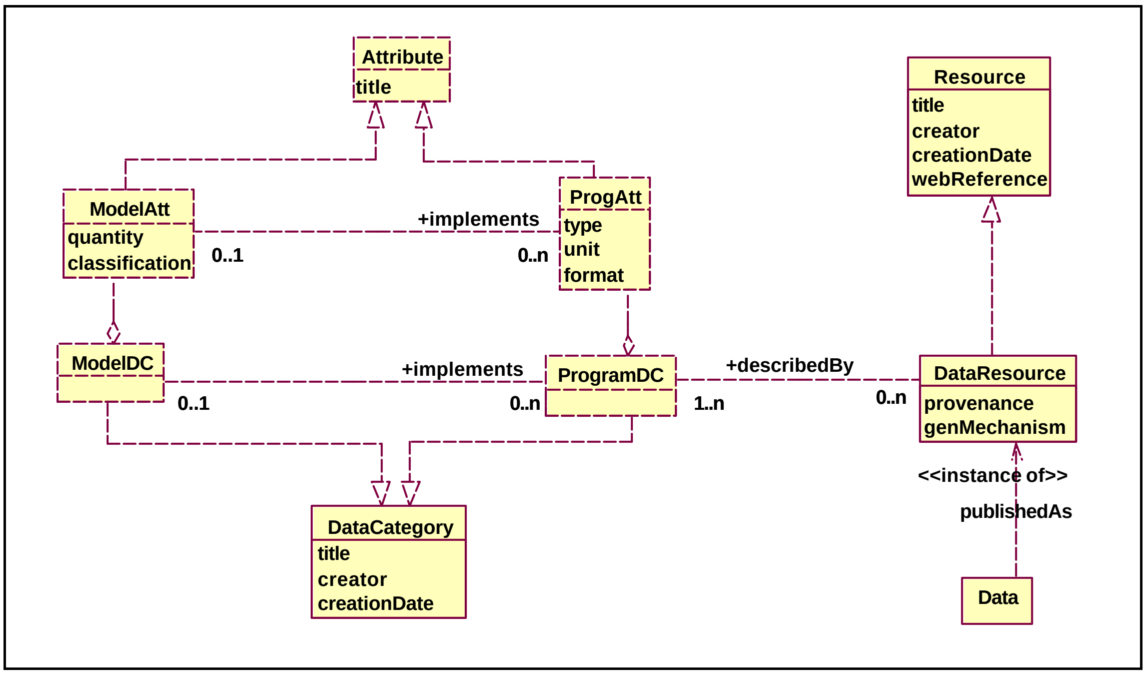


Figure 38: SPM data category related concepts

When a program implements a model, this relationship is extended to the associated data categories. Therefore, a mapping function should exist between a program and a model, meaning that for each I/O data associated to the model, there must be a related I/O data associated to the program. A valid implementation relationship should map each and all I/O data associated to the model to one unique correspondent I/O data associated to the program, i.e., the mapping function between data categories associated to both model and program should be an injective function.

Analogously, we may say that a program data category implements a model data category, by establishing a mapping function between both data categories. This mapping means that each attribute of the model data category must be related to one attribute of the program data category. A valid implementation should relate each and all attributes of the model data category to one unique attribute of the program data category, i.e., the mapping function between attributes from the model to the program data categories should be an injective function.

A **parameter** is a concept that can represent: (i) a *model parameter* which is used to "tune" the model for a specific objective (e.g., one may say that in the formula "**y=ax+b**" that represents a "straight line", **a** and **b** are *constant parameters* which determine the intersection points between the line and the axes); (ii) a *processing parameter* which is used to determine some "performance" and "accuracy" aspects (e.g., a program requiring a minimum set of data, or even, a program using alternative precision options, according to user needs); (iii) a *condition parameter* which is created only to express a value required by some program use constraint (e.g., a program that should be used only for a certain data value interval).

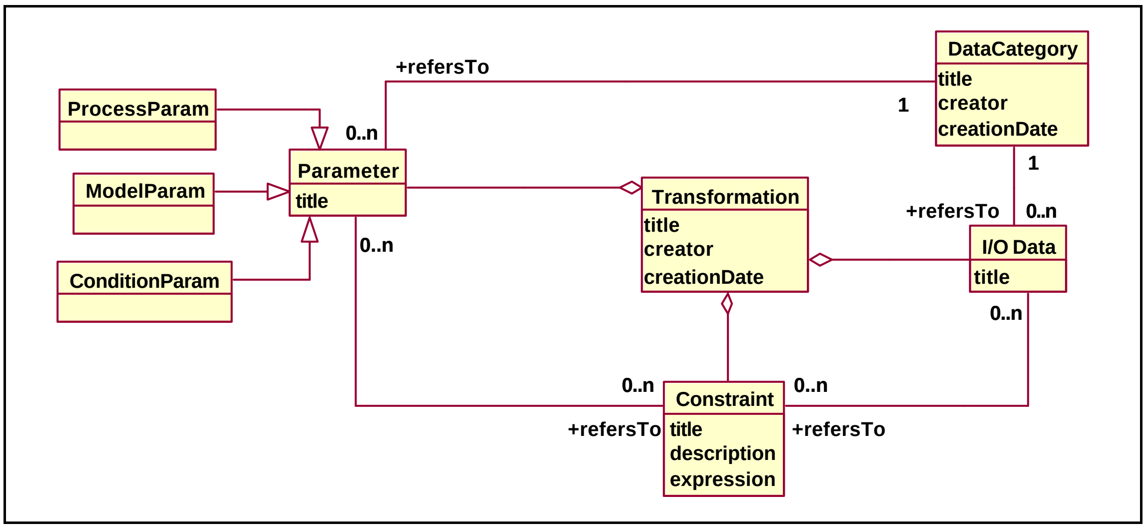


Figure 39: SPM parameter and constraint concepts

As Figure 39 shows, a parameter refers to a data category that might be at the program or model level. The user can either define a data category for each parameter, or define a set of parameters under the same data category, as its attributes. However, it is important to notice that the user should have in mind which data is really a transformation data input, and which is clearly defined as a parameter.

Constraints describe conditions for the adequate use of a transformation. When describing a model or program it is important to express its limitations, i.e., which constraints it should be conformed to, in order to be used correctly. A program is itself a computational model of a theoretic model. Precision and performance are some of the issues that constrain a program. Examples of some common transformation constraints are: (i) a model developed for a specific geographic location, should be constrained to data from that location (e.g., an image segmentation model that has been designed only for images from a specific region); (ii) a program developed to run over some data input cardinality lower-bound (e.g., a data simulation program that would run based on a minimum historic data interval); (iii) a model developed to run specifically over some data value interval (e.g., a biocorrosion model that would fit only to water samples whose chloride concentration is below a given value).

As shown in Figure 39, a constraint might be associated to I/O data and/or to parameters, depending on the data involved in the constraint. Each constraint is characterized by **description** and **expression** attributes, which represent informal and formal constraint expressions, respectively. In the case of formal descriptions, it is recommended to use some formal language.

In summary, all concepts discussed here are represented in the SPM metamodel. Figure 40 presents a simplified overview of the SPM metamodel, which shows those concepts and their relationships.

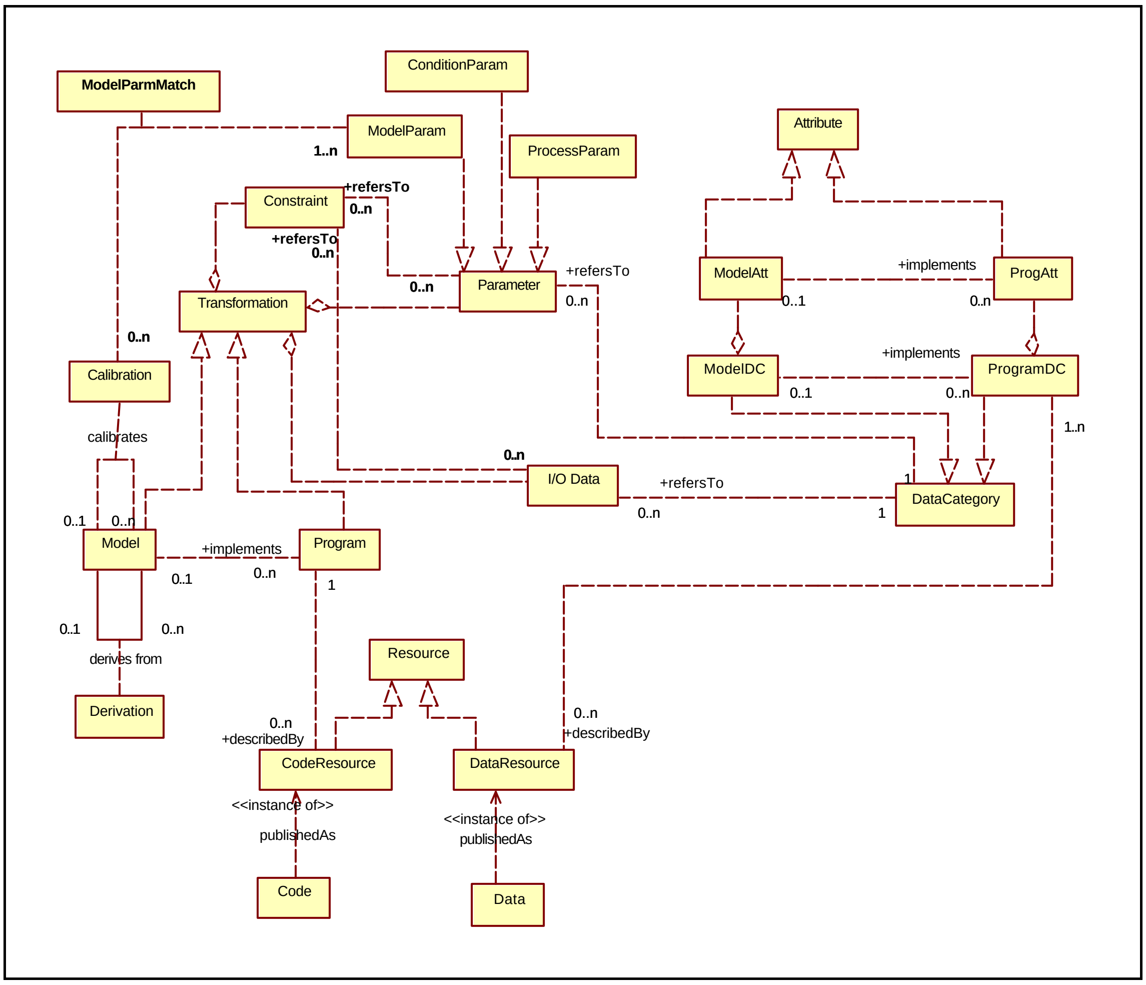


Figure 40: SPM Schema overview

4.2.2 SPM Advanced Concepts

Other advanced concepts must be identified to provide for the resource usage registry. Figure 41 presents these advanced concepts and their relationships to the other original concepts.

The transformation concept is used to describe a transformation resource, such as models and programs. However, to make a transformation available for experiments, we use another concept: **workflow**. Therefore, a **transformation** that needs to be available for experiments should be declared as part of a **workflow**. A workflow is related to a set of transformations and is described by a **workflow specification** attribute, which should contain a formal workflow description language like in BPEL4WS (CURBERA, F., GOLAND, Y, *et al.*, 2002). Such workflow specification describes how the related transformations are to be processed, i.e., which transformation should be performed first, which others may be performed in parallel, which transformation output data corresponds to other transformation input data, etc.

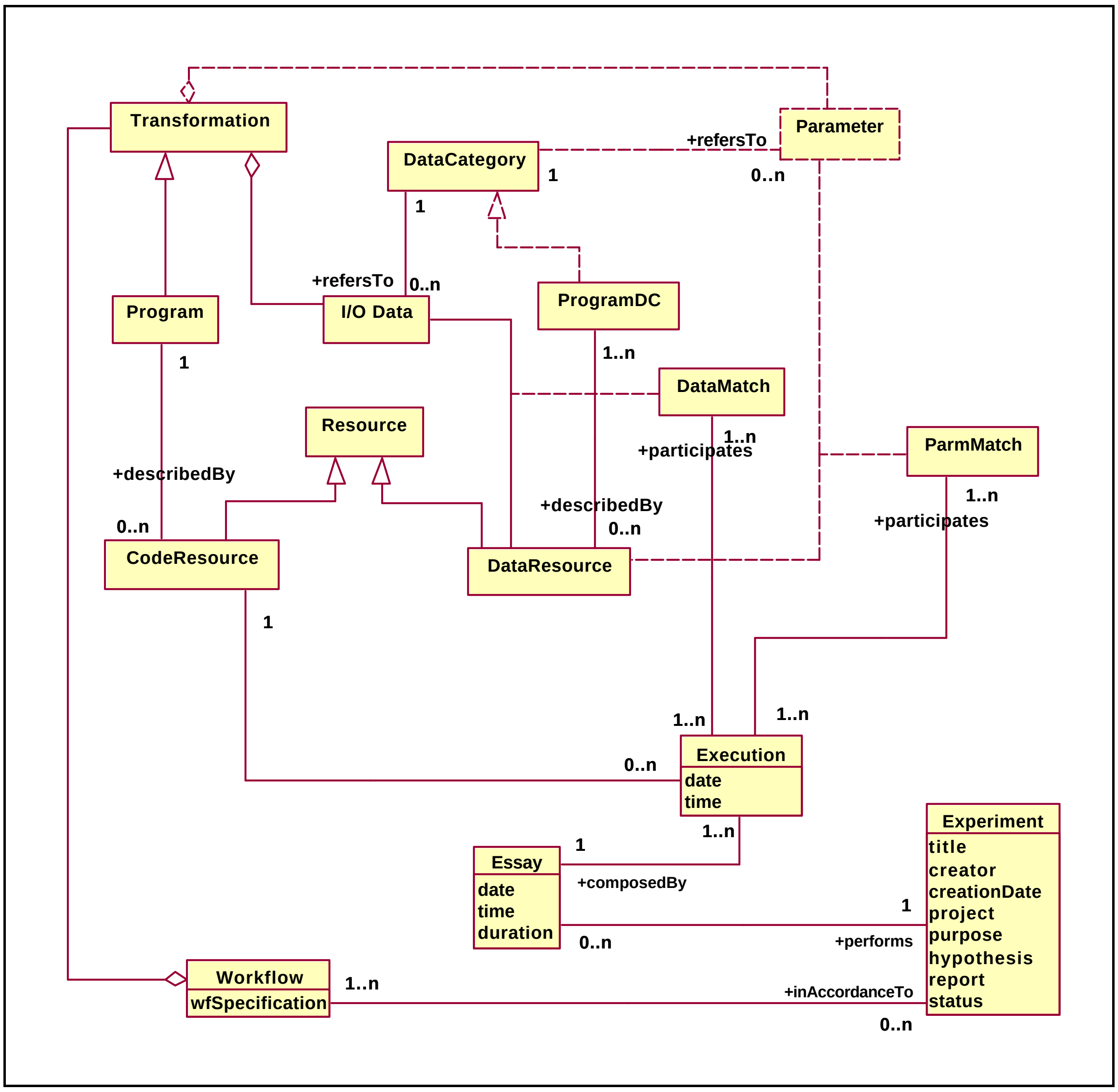


Figure 41: SPM advanced concepts (experiment and workflow)

In the case of our *in silico* laboratory environment the definition of experiment presented in section 2.3.5 fits well. Each *in silico* **experiment** has its own hypothesis and purpose. The control is established through the association to a set of related workflows, i.e., an experiment has a fixed set of workflows over which actions are taken. Each action begins with a workflow instantiation, and ends when the workflow execution is complete. To use a common lab word, each workflow instantiation is what is called an **essay**. An essay involves a set of code executions. These executions correspond to an instantiation of a specific workflow, which is composed of a set of programs.

A code **execution** describes each use of a program by keeping a record of which resources (code and data) the scientist used during an essay, i.e., the code **execution** registers for each program parameter, which value was used (**parmMatch**) and for each program I/O data, which data resource was used (**dataMatch**). A **data match** is the assignment of a data resource to a data I/O that belongs to a program interface. The assignment process should verify the compatibility among the data categories referred by both data resource and program input data. The data input assignment happens during the workflow instantiation, while the data output assignment happens at runtime. In summary, the code **execution** registers the use and generation of data resources as data I/O of a code resource, during an *in silico* essay.

4.3 Analysing SRM in the light of the requirements

Figure 42 shows that SRM architecture provides most of the facilities listed in the table of Figure 26. In the first group of facilities, SRM does not attend a complete execution control. However, we believe it is not difficult to add this facility, especially if SRM is used in combination with OGSA web services. Differently from the other initiatives (MyGrid, GryPhin, ESSW, ESP2Net, etc.), SRM strongly concentrates on the description facilities, attending most of them. In special, we have seen the importance of the differentiation between models and programs in almost all applications we analysed in section 2, particularly, in the biocorrosion application (SIMBIO project). The only work that considers this difference (ECOBAS-MIF) does not adopt a metamodel approach. Besides, ECOBAS-MIF does not include important scientific concepts such as essay and experiment that represent the use of models and programs. We were able to verify the use of these concepts in a structural genomic application (MHOLline project), analysed in section 2. Although SPM metamodel is more expressive than its related works, it did not include the concept of data replica. However, we believe SPM can be easily extended to add this concept. Thus, SPM metamodel contributes by providing rich modelling concepts that impact on almost all activities of the e-scientist, including concepts that were not considered in other related works so far.

The workflow dynamic definition facility is not yet attended. To attend this facility, the SPM metamodel should include workflow versioning control, which is out of the scope of this work. However, we believe it would be an interesting future work direction.

	Distribution and Heterogeneity facilities
✍	Data Distribution
✍	Data Heterogeneity
✍	Program Distribution
✍	Program Heterogeneity
✍	Prog. remote execution
Some	Execution control
	Description facilities
✍	Model description
✍	Program description
✍	Program IO description
✍	Data Type description
✍	Code description
✍	Data description
✍	Data replica description
✍	Description Extensibility
	Wf and registry facilities
✍	Wf definitions
✍	Abstract wf definitions
✍	Wf dynamic definition
✍	Wf execution
✍	Wf partial execution
✍	Code execution registry
✍	Wf instance registry
✍	Experiment registry

Figure 42: Requirements review for SRM

Finally, it is worth to mention that SRM adopted Web services as the technology for handling distribution and heterogeneity of scientific resources. The next section describes SRM implementation, and evidences how Web services are a promising and suitable technology.

5. Web Services based SRM architecture Implementation

Web services are the standard technology to address interoperability issues, thus SRM architecture benefits from its infrastructure, summing its functionality atop Web Services. In 2001, after specifying SRM we developed prototypes using Le Select as a middleware system, which uses CORBA for interoperability. At that time, we proposed SRM as an extension of Le Select (CAVALCANTI, M. *et al.*, 2002), and extended the ODBMS GOA (MATTOSO, M., CAVALCANTI, M., *et al.*, 2002) to manage XML documents. In parallel, we were able to study SIMBIO project application and identify SRM contributions to this context (ECOBASE, 2001) (CAVALCANTI, M. *et al.*, 2002 c). However, in 2002, Web services emerged as a promising standard technology. Despite the advantages of Le Select, in general, the scientific community is very standard oriented and (especially in the case of biologists) rely only on freeware, shareware or openware software, which is not the case with Le Select. For that reason, SRM architecture is currently proposed as a Web services based architecture (CAVALCANTI, M., MATTOSO, M., CAMPOS, M. L., 2003 a).

In this section we present the Web services based Scientific Management Architecture (SRMW). Section 5.1 details each SRM module with respect to implementation issues. In addition, we also present the SPMW metamodel expressed in XML Schema and how it extends WSDL.

5.1 SRMW Architecture Implementation

SRMW modules are presented in Figure 43. The Web services provider corresponds to the Resource Operation module of the SRM architecture. A Web services provider can serve both data and code as services. To facilitate the search for these services, the publisher registers (step 1) his/her services through the Publication module, extending the correspondent WSDL documents with SPMW description elements. These documents are stored and managed by a Web services registry facility, adapted to scientific requirements, which corresponds to the SRM Resource Description module.

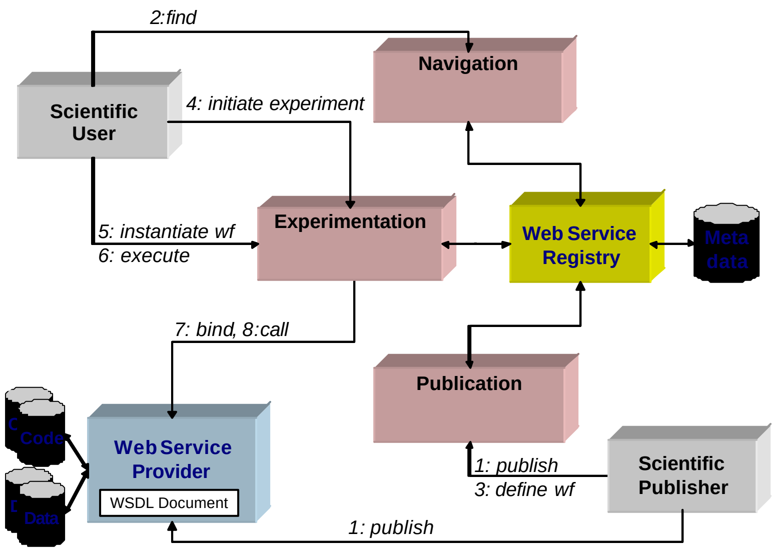


Figure 43: SRM Architecture based on Web services (SRMW)

SRMW navigation module helps to find a service (step 2) by guiding the user through a collection of WSDL extended documents. As the user finds what it seems to be appropriate, the Publication module helps on planning an orchestrated execution (step 3) through a workflow definition. Then, this workflow definition is associated to a scientific experiment which must be registered through the Publication module (step 1). This experiment is then initiated (step 4), and the user is now able to start the instantiation of the defined workflow (step 5) and its subsequent execution (step 6). The Experimentation module acts like a Web services requestor (steps 7 and 8) connecting to the required Web services providers, which will actually access data sources and execute the service codes.

5.1.1 Resource Operation Module

Data and code publishers are responsible for building a Web services provider for their resources, so that they can become available for Web users. For each legacy code the user should build a Web services adapter (code wrapper). The generic invocation protocol adopted by the Web services provider may be SOAP, which would be processed by a SOAP router. In this case, the code requestor of the Experimentation module would send SOAP messages to the Web services provider, as service client requestor. Analogously, for each data set there must be a Web services adapter (data wrapper).

Building code and data Web services, for the scientific community, is not an easy task. This is an open issue in Web services based architectures like SPMW, and out of the scope of this work. So far, only generic data access Web services have already been proposed. The OGSA Data Access and Integration (OGSA-DAI, 2003) is one of the main initiatives in this direction. A master's dissertation to facilitate the generation of scientific data resources Web services is under development (TEIXEIRA, F., 2003) at COPPE Sistemas.

In addition, the investigation of the difficulties on building a biological workflow using Web services is the focus of another master's dissertation work (TARGINO, R., 2003) at COPPE Sistemas. In this context, we have published Web services for some bioinformatics legacy codes (CAVALCANTI, M. *et al.,* 2003 b), using Apache Tomcat 4.0.4 (TOMCAT) powered by the AXIS engine (AXIS)

5.1.2 Resource Description Module

The Resource Description module corresponds to an adapted Web services Registry. It registers descriptions according to the SPMW metamodel. To accommodate all metadata captured by this metamodel, we use a database server as RD main component. As a metadata repository manager, it stores resource descriptions expressed in XML. XML-enabled RDBMS and native XML databases are both alternatives for XML storage. We have investigated two alternatives, building the RD module on top of GOA and MySQL Systems.

Initially, due to the similarity between the object oriented model and the XML model, we have chosen to store XML documents in an ODBMS. We have XML enabled the GOA System (GOA SYSTEM), an ODBMS prototype developed at COPPE Sistemas. We have embedded not just a GOA Client, but also the GOA XML enabler (*Goaxe* API) facility (MATTOSO, M., CAVALCANTI, M., *et al.,* 2002). With *Goaxe*, the GOA System is able to understand and store XML documents. *Goaxe* manipulates XML documents by creating a GOA XML Schema that reflects the W3C DOM API classes. *Goaxe* takes an XML document instance, reads it and breaks it down into DOM class instances. Then, each of these instances is translated into a GOA XML Schema instance. Therefore, the GOA Server can be viewed as a generic XML repository. In addition, through XVerter (VIEIRA, H., RUBERG, G., MATTOSO, M., 2002) XQuery commands can be issued to GOA stored XML documents.

However, due to the wide use of MySQL on the Web, as well as on some scientific communities, we have changed the implementation of metadata repository to the RDBMS MySQL (WIDENIUS, M., AXMARK, D., 2002). Any repository can be used, as long as it can offer an XML facility. To provide XML access to MySQL, we have extended an XML open source API (NIEMCZYK, B., 2002) and associated it to a MySQL client API. The XML API breaks any XML document into four relations, also acting as a generic XML repository.

The Indexer module was not yet implemented. In the case of a native XML DBMS, this module may already exist.

The metadata repository stores also WSDL documents, extending the original ones with imports pointing to the SPMW extensions. Other Web applications may benefit from these extended WSDL documents by connecting directly to the Resource Description module.

5.1.3 Web Access Layer Implementation Issues

The Publication module is currently under implementation using Java Servlets, interfacing with any Web browser through HTML pages and forms. These servlets connect to MySQL server through a PHP client API. The XML API used was also written in PHP. Each Java Servlet manipulates XML documents using the Apache Project DOM API (XERCES), which was implemented based on the W3C DOM API specification. The idea is to deliver an XML valid document to the Resource Description module, ready to be stored by the DBMS.

The Schema Processor facility is a Java Servlet that is called for each scientific element publication. It queries the RD module to get a subset of the schema according to the type of scientific element under publication. The schema subset includes enumeration elements derived from the XML documents to be referenced, e.g., programs should reference model descriptions. The Interface Builder automatically transforms the edited schema into a publication interface. The Interface Builder was implemented in XML Stylesheet Language (XSL) as a final under-graduation course project (LEAL, L.N., 2003). The generated interface includes validation routines according to the domain constraints defined in the original schema, such as pattern and interval validations.

The Navigation module is under development. It also comprises a set of Java Servlets. So far, we have implemented the Navigation interface, which allows the user to browse XML documents according to its category in SPMW model. The keyword search facility is not yet implemented. The Schema Processor facility is under development, and the Interface Builder is being developed as part of a final under-graduation course project.

The Publication and Experimentation modules count on IBM BPWS4J 1.0.1 (BPWS4J) to define and process workflow specifications, written in BPEL4WS (CURBERA, F., GOLAND, Y, *et al.*, 2002). A tool for the Experimentation module is being developed as part of a master's dissertation (TARGINO, R., 2003), based on a prototype evaluated with MHOLline (CAVALCANTI, M. *et al.*, 2003 b). Integration between this tool and SRMW is planned. For example, BPWS4J edition facilities could benefit from the selection of programs during the user navigation through scientific resources. Then, the generated workflow definition documents might be returned to the WF Definition Validation facility to be validated and, subsequently, stored in the metadata repository.

The Experimentation module is also under development. A set of Java Servlets will be interacting with the user and the RD module to start a new essay. The instantiation of the workflow is then started, and the definition of code/data selection criteria defines the complete workflow instance, which is finally available for execution. Then, the WF Engine facility resolves any pending choice according to the user defined criteria and processes the workflow instance, which is composed by a set of Code and Data Web services requests. For each request, the requestor module builds on the fly SOAP messages.

As part of a master's dissertation work (TARGINO, R., 2003), a static scientific workflow has been implemented (CAVALCANTI, M. *et al.*, 2003 b), using Web services technology. We have published legacy code using WSDL based on SPMW and used BPWS4J as the workflow engine, and data results are being stored locally, as XML pages. SRMW Experimentation module is now under integration with this work.

5.2 SPMW Implementation

To take advantage of the Web services technology we have implemented our metamodel as an extension to WSDL. Considering that services and programs are equivalent concepts, both metamodels (SPMW and WSDL) share the same objective. Therefore, it is natural to find some intersection between them. Such intersection involves mainly program description elements. In fact, SPMW complements WSDL providing more descriptive elements and relationships. In order to be standard compliant we have chosen to extend WSDL metamodel with SPMW semantic elements, as illustrates Figure 44.

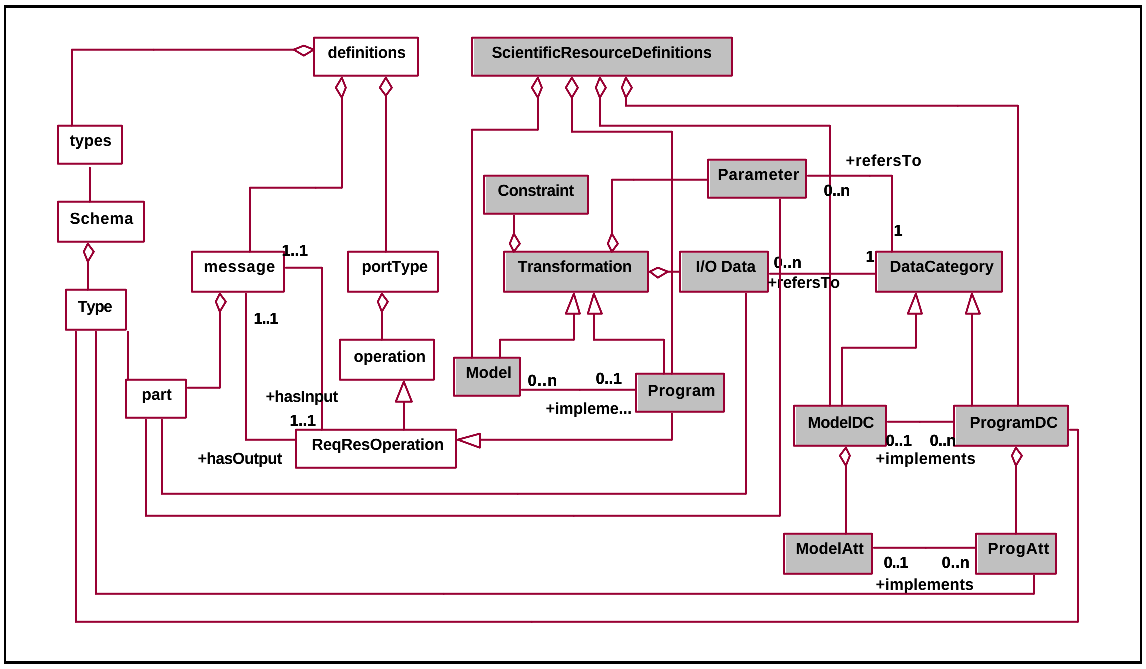


Figure 44: WSDL and SPMW mapping (abstract part). SPMW elements are painted.

Analogously to WSDL, we have created a new element that is composed by SPMW definitions, called `ScientificResourceDefinitions`. Each scientific resource, ie., program, model, program data categories, etc., must be declared under this element. The `program` element extends the `ReqResOperation`. As a consequence, each `DataIO` and `Parm` related to `program` should refer to the correspondent `part` of the `ReqResOperation`, and each `ProgramDC` (and its attributes – `ProgAtt`) referred by `DataIO` and `Parm` elements should refer to a `Type` declared within the `types` element. These relationships are represented in the SPMW XML Schema (presented in the next section), as extra attributes whose values should point to WSDL element instances.

Another mapping happens between WSDL and SPMW at the concrete level. Code and data resources are also SPMW elements defined under the `ScientificResourceDefinitions`. Both elements provide information about code and data hosts, complementing the access and encoding information provided by each binding at a WSDL document. In Figure 45, the diagram shows that `CodeResource` and `DataResource` elements are connected to the `bindingOperation` element.

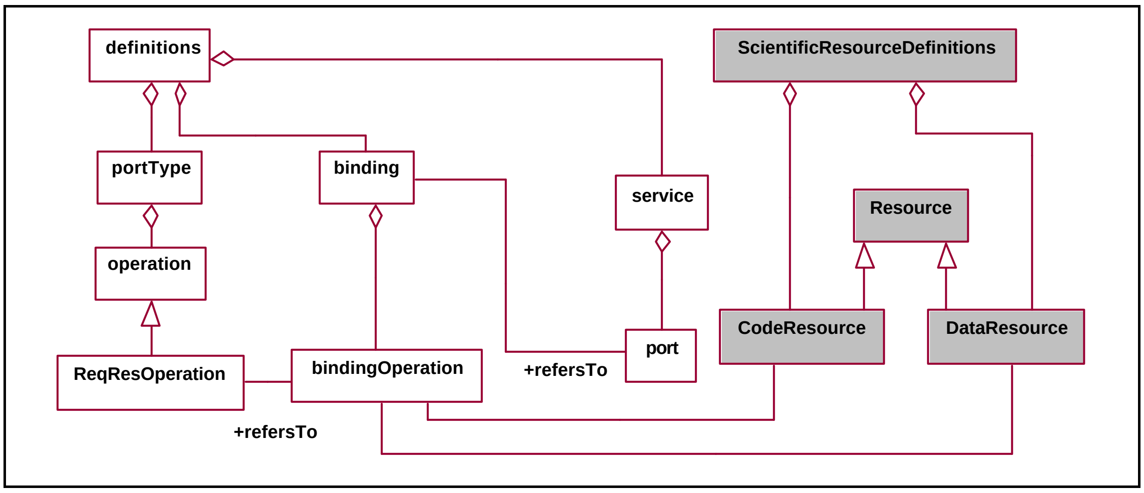


Figure 45: WSDL and SPMW mapping (concrete part). SPMW elements are painted.

5.2.1 SPMW XML Schema

SPMW XML Schema has as its root the `ScientificResourceDefinitions` element. According to this schema, an SPMW document may include any number greater than zero of scientific resources elements under this element, in any order, as illustrated in Figure 46.

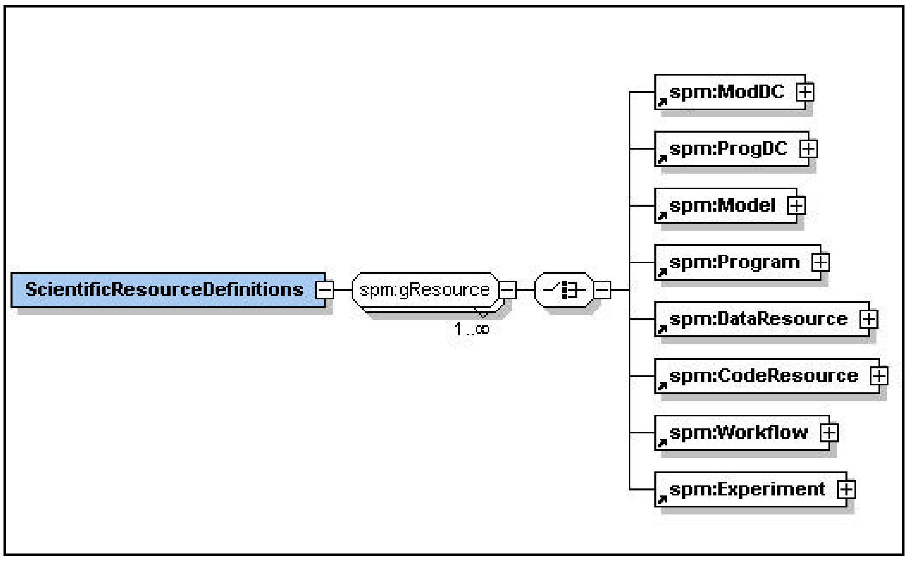


Figure 46: SPMW main elements

These eight elements are detailed in the next sub-sections. Hierarchic diagrams are used to present each of these elements in separate, while the whole XML Schema is available in the Appendix. Some elements are optional, and appear in dotted boxes. Some elements are extensions of abstract elements, taking advantage of the inheritance mechanism available in XML Schema design. The inherited elements appear first as a separate group of elements. Each of the main elements includes an extensibility element (`any` element), allowing new sub-elements to be included in instance documents. Elements that have a complex structure are indicated by the plus (+) sign.

`Key` definitions are used to provide unique identifications to instances of SPMW elements. `Keyref` definitions are used to establish the relationships between elements within SPMW, and consequently, to provide referential integrity among the correspondent instances. In the case of references between SPMW and WSDL elements, `key` and `keyref` definitions are not used. The reason of choosing a less tight relationship between these schemas is to provide more independence to SPMW, enabling it to be associated to other schemas.

SPMW XML Schema current version is not fully addressing the SPMW conceptual model. Some aspects were left out and will be developed along with application projects, while SPMW evolves.

5.2.1.1 ***Model*** – spm:Model

The model element inherits some attributes from the abstract type `tTransformation`, which are included in the first group of elements in Figure 47. `Title`, `creation` and `creationDate` are self-explanatory elements. `Input`, `parm` and `output` are elements that may occur many times. All three of them have a similar complex structure that includes a `title` and a `reference` to a data category. The `constraint` element also has a complex structure. However, this one is different and includes three elements: a `title`, a `description`, used to describe it in natural language, and an `expression`, used to describe it in a formal language.

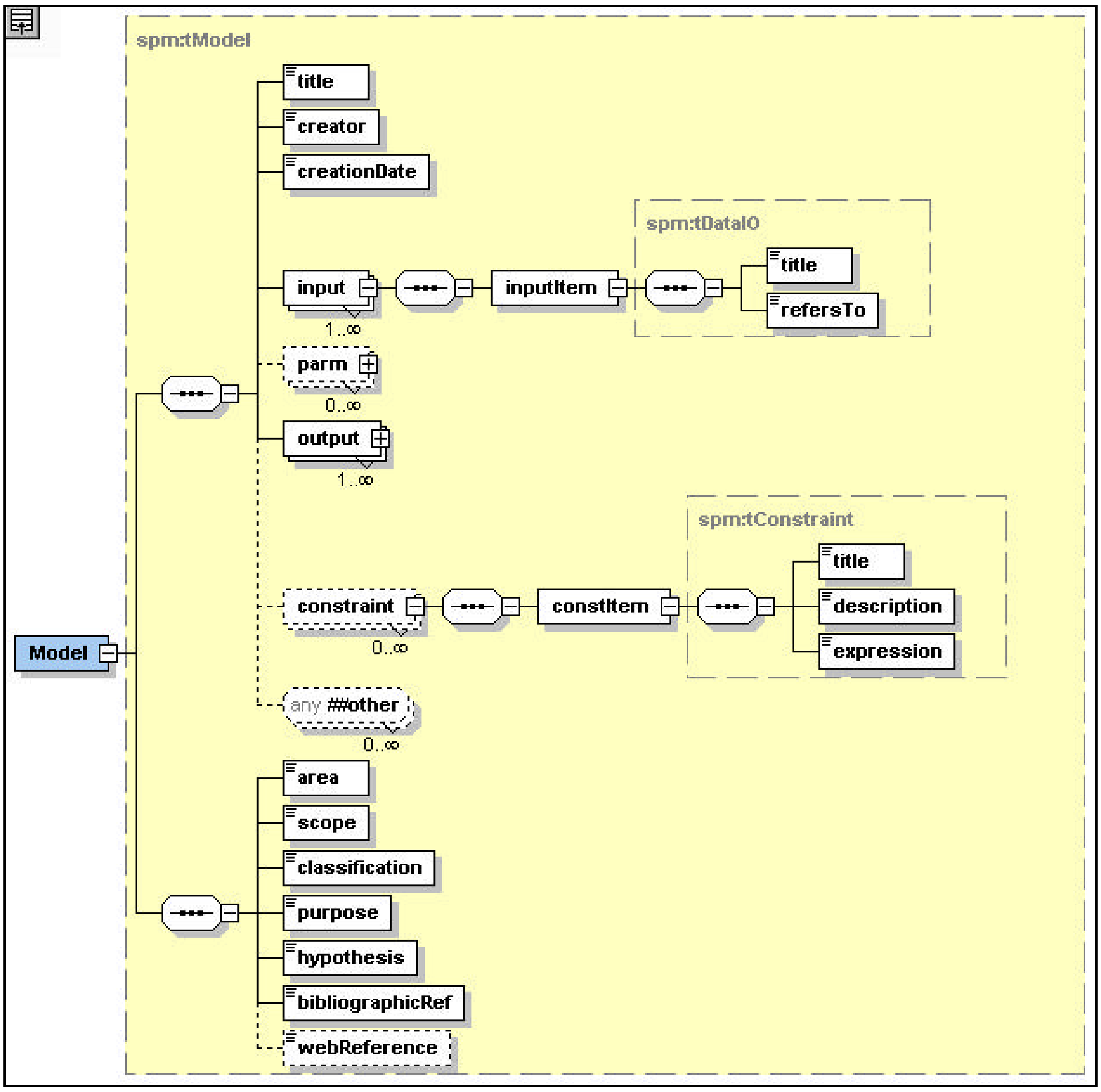


Figure 47: Model XML Schema diagram

In the second group of elements there are model specific descriptive elements:

?? `Area:` A model is usually associated to an area of application, e.g. industrial, economic, social, political, environmental, etc.

?? `Scope:` The target or scope of a model is the system it represents, e.g. Itajaí hydrographic basin, a geographic region, or an enterprise.

?? `Classification:` There are many different ways of classifying a model, e.g. mathematic, logic, deductive, empiric, probabilistic, algorithmic, simulation, etc.

?? `Purpose:` Each model has a specific purpose, for which it is valid.

?? `Hypothesis`: Every model is initially a hypothesis. Building a model represents the expression of a scientific hypothesis that needs to be validated.

?? `BibliographicRef`: A model is usually associated to scientific publications.

?? `WebReference`: It would be useful to register the Web address of the model reference material, if it exists.

As shown in Figure 48, the `model` element has a unique attribute that represents its key (`kModel`). `Input`, `parm` and `output` elements include the element `refersTo`, which is committed to `ModDC` element, through a `keyref` definition.

```
<xsd:element name="Model" type="spm:tModel">
    <xsd:key  name="kModel">
        <xsd:selector xpath=”.”/>
        <xsd:field xpath=”@idTF”/>
    </xsd:key>
    <xsd:keyref name=”refInputModDC” refer=”spm:kModDC”>
        <xsd:selector xpath=”input”/>
        <xsd:field xpath=”refersTo”/>
    </xsd:keyref>
    <xsd:keyref name=”refOutputModDC” refer=”spm:kModDC” >
        <xsd:selector xpath=”output”/>
        <xsd:field xpath=”refersTo”/>
    </xsd:keyref>
    <xsd:keyref name=”refModParm” refer=”spm:kModDC”>
        <xsd:selector xpath=”parm”/>
        <xsd:field xpath=”refersTo”/>
    </xsd:keyref>
</xsd:element>
```

Figure 48: Model Key and Keyref definitions

5.2.1.2 ***Model Data Category*** – spm:ModDC

The `ModDC` element represents the Model Data Category concept. It inherits from `tDataCategory` abstract type some descriptive self-explanatory elements: `title`, `creator` and `creationDate`, which are included in the first group of elements in Figure 49. `ModDC` also includes another group of elements which may contain multiple occurrences of `MDCAttribute` element. Each occurrence of this element is described by its `title` and a related `quantity`. A previously prepared list of quantities may be available.

As shown in Figure 50, `ModDC` and `MDCAttribute` elements have a unique attribute that represents their key (`kModDC` and `kModAttribute`).

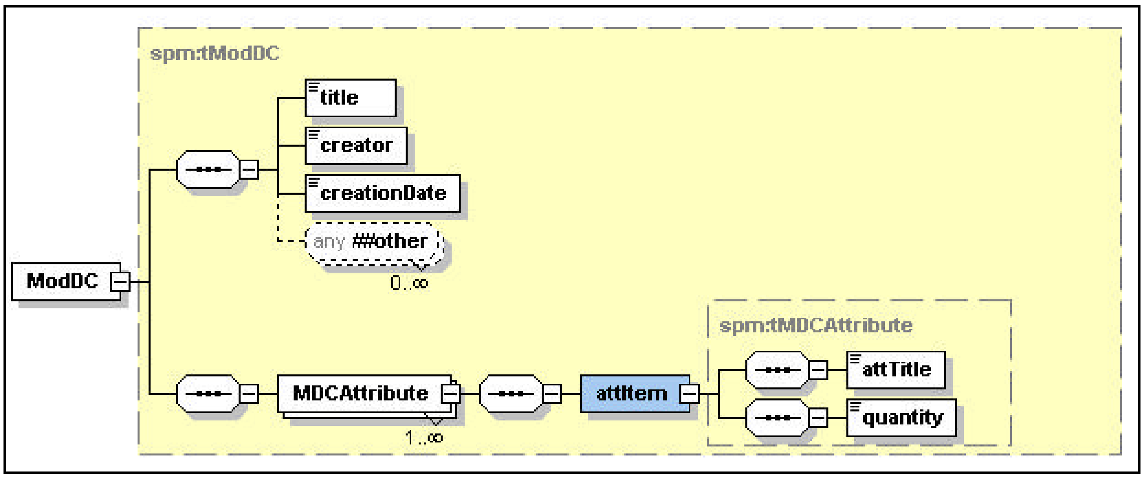


Figure 49: Model Data Category XML Schema diagram

```
<xsd:element name=”ModDC” type=”spm:tModDC”>
    <xsd:key  name=”kModDC”>
        <xsd:selector xpath=”.”/>
        <xsd:field xpath=”@idDC”/>
    </xsd:key>
</xsd:element>
<xsd:element name=”ModAttribute” type=”spm:tMDCAttribute”>
    <xsd:key  name=”kModAttribute”>
        <xsd:selector xpath=”.”/>
        <xsd:field xpath=”@idAttribute”/>
    </xsd:key>
</xsd:element>
```

Figure 50: Model Data Category Key definition

5.2.1.3 ***Program*** – spm:Program

The `program` element inherits some attributes from the abstract type `tTransformation`, which are included in the first group of elements in Figure 51. These elements were already discussed in section 5.2.1.1. The `constraint expression` element is used to describe a program constraint in a formal language, possibly using BPEL4WS.

In the second group of elements there are program specific descriptive elements. `ImplementationLanguage` is the programming language with which the program was implemented. It might be important to specify the version of this language. The `Version` element is used to specify the version/release of the program under description. Finally, the `wsdlElementRef` represents the correspondence to a port type operation of a WSDL document.

As shown in Figure 52, the `program` element has a unique attribute that represents its key (`kProgram`). `Input`, `parm` and `output` elements include the element `refersTo`, which is a reference to `ProgDC` element instances, through `keyref` definitions (`refInputProgDC`, `refParmProgDC`, `refOutputProgDC`). Another `keyref` definition involves the `implements` element, which is used to make reference to the model implemented by a program (`refModel`).

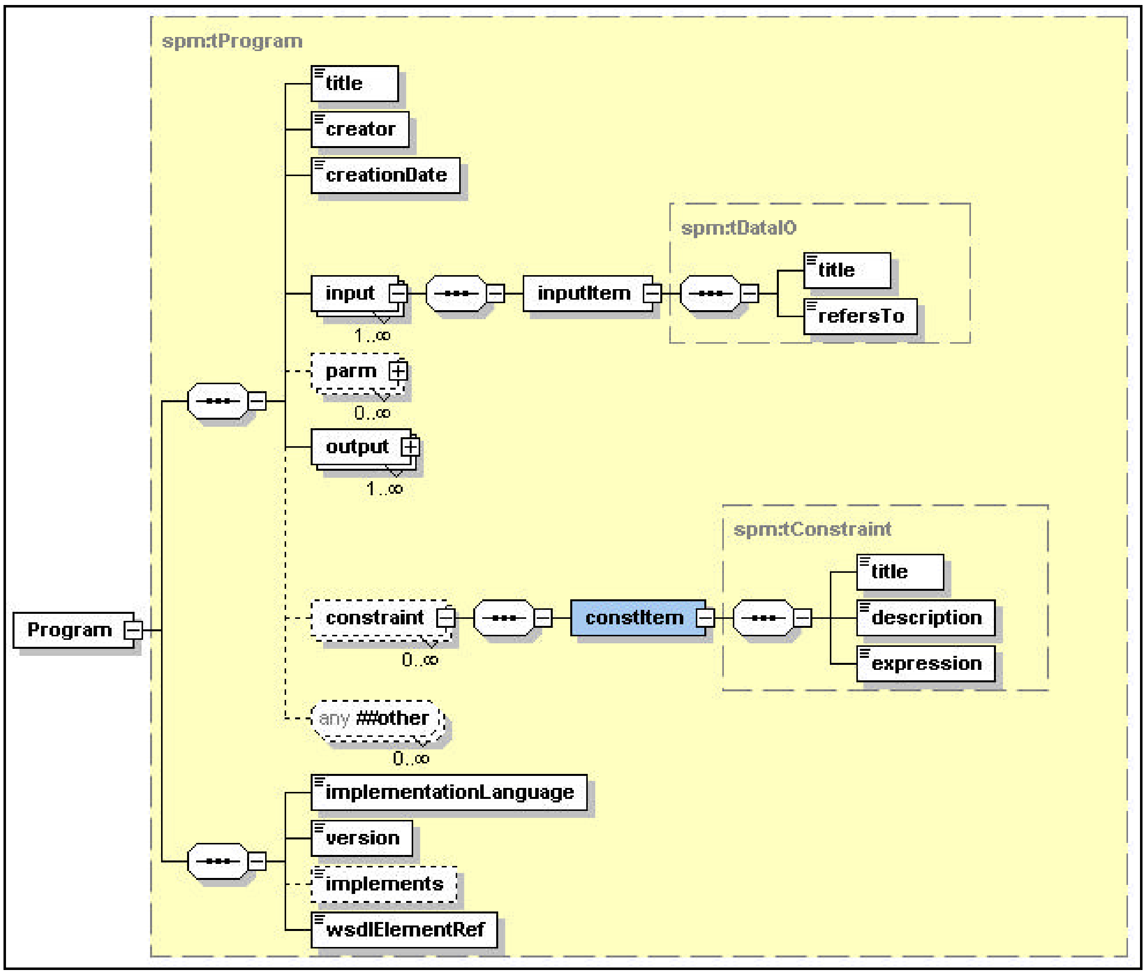


Figure 51: Program XML Schema

```
          <xsd:key  name="kProgram">
              <xsd:selector xpath="."/>
              <xsd:field xpath="@idTF"/>
          </xsd:key>
          <xsd:keyref name="refModel" refer="spm:kModel">
              <xsd:selector xpath="."/>
              <xsd:field xpath="implements"/>
          </xsd:keyref>
          <xsd:keyref name="refInputProgDC" refer="spm:kProgDC">
              <xsd:selector xpath="input"/>
              <xsd:field xpath="refersTo"/>
          </xsd:keyref>
          <xsd:keyref name="refOutputProgDC" refer="spm:kProgDC">
              <xsd:selector xpath="output"/>
              <xsd:field xpath="refersTo"/>
          </xsd:keyref>
          <xsd:keyref name="refProgParm" refer="spm:kProgDC">
              <xsd:selector xpath="parm"/>
              <xsd:field xpath="refersTo"/>
          </xsd:keyref>
      </xsd:element>
```

Figure 52: Program Key and Keyrefs definitions

5.2.1.4 *Program Data Category–* spm:ProgDC

The ProgDC element represents the Program Data Category concept. It inherits from tDataCategory abstract type some descriptive self-explanatory elements: title, creator and creationDate. These elements are included in the first group of elements in Figure 53. In a second group of elements ProgDC also includes some specific elements. The PDCAttribute element may have multiple occurrences, and each occurrence is described by its title and a related unit. A previously prepared list of units may be available. The wsdlElementRef element, which appears within ProgDC and PDCAttribute elements, makes reference to a part and type in a WSDL document, respectively.

As shown in Figure 54 , ProgDC and PDCAttribute elements have a unique attribute that represents their key (kProgDC and kProgAttribute). The implements element, which also appears within ProgDC and PDCAttribute elements, is used to make reference to model level elements instances (ModDC and MDCAttribute), through keyref definitions (refModDC and refModAttribute).

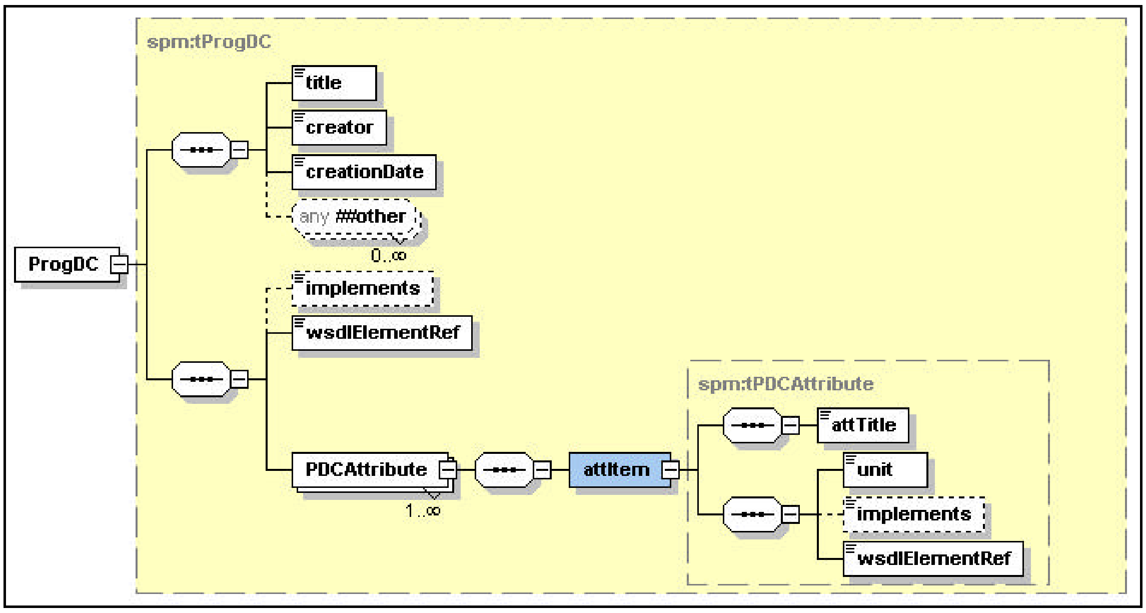


Figure 53: Program Data Category XML Schema diagram

```
<xsd:element name=”ProgDC” type=”spm:tProgDC”>
    <xsd:key  name=”kProgDC”>
        <xsd:selector xpath=”.”/>
        <xsd:field xpath=”@idDC”/>
    </xsd:key>
    <xsd:keyref name="refModDC" refer="spm:kModDC">
        <xsd:selector xpath="."/>
        <xsd:field xpath="implements"/>
    </xsd:keyref>
</xsd:element>
<xsd:element name="ProgAttribute" type="spm:tPDCAttribute">
    <xsd:key  name="kProgAttribute">
        <xsd:selector xpath="."/>
        <xsd:field xpath="@idAttribute"/>
    </xsd:key>
    <xsd:keyref name="refModAttribute" refer="spm:kModAttribute">
        <xsd:selector xpath="."/>
        <xsd:field xpath="implements"/>
    </xsd:keyref>
</xsd:element>
```

Figure 54: Program Data Category Key and Keyrefs definitions

5.2.1.5 *Code Resource*– spm:CodeResource

The `CodeResource` element represents a code, wrapped by a Web Service, hosted by a particular computer in a specific address. As shown in Figure 55, it inherits a group of descriptive self-explanatory elements from the `tResource` abstract type: `title`, `creator` and `creationDate`. The `describedBy` element in the same group associates the code resource to its program interface description. Finally, the `wsdlElementRef` is used to make reference to a port operation in a WSDL document.

Another group of elements describes specific characteristics to a code resource: `hardwareInfo` and `operationalSystem`. When about to decide which remote code to execute, the hardware and operational system are useful information to consider as selection criteria, among others that eventually will be added.

As shown in Figure 56, the `codeResource` element has a unique attribute that represents its key (`kCodeResource`). The `describedBy` element is used to reference `program` element instances, through a `keyref` definition (`refDescribedByProgram`).

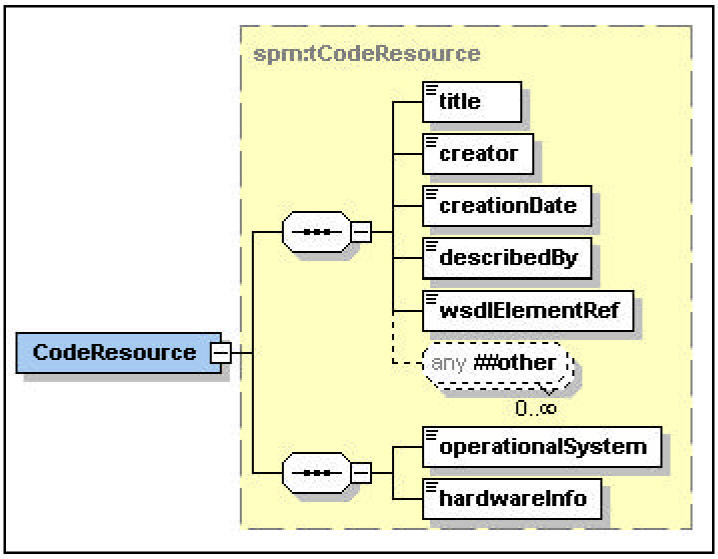


Figure 55: Code Resource XML Schema diagram

```
<xsd:element name="CodeResource" type="spm:tCodeResource">
    <xsd:key name="kCodeResource">
        <xsd:selector xpath="."/>
        <xsd:field xpath="@idCR"/>
    </xsd:key>
    <xsd:keyref name="refDescribedByProgram" refer="spm:kProgram">
        <xsd:selector xpath="."/>
        <xsd:field xpath="spm:describedBy"/>
    </xsd:keyref>
</xsd:element>
```

Figure 56: Code Resource Key and Keyrefs definitions

5.2.1.6 ***Data Resource*** – spm:DataResource

The `DataResource` element represents a data set, wrapped by a Web Service, hosted by a particular computer in a specific address. As shown in Figure 57, it inherits a group of descriptive self-explanatory elements from the `tResource` abstract type: `title`, `creator` and `creationDate`. The `describedBy` element in the same group associates the data resource to its program data category description. Finally, the `wsdlElementRef` is used to make reference to a port operation in a WSDL document.

Another group of elements describe specific characteristics to a data resource: `provenance`, `genMechanism` and `WebReference`. When about to decide which remote data set to access, the data generation mechanism (e.g. if they were produced by a satellite, sensor, program, etc.) and data provenance (i.e. the identification of the source that had generated it) are useful information to consider as selection criteria. The `WebReference` element is used as an alternative for accessing data sets which are already available in XML format.

As shown in Figure 58, the `dataResource` element has a unique attribute that represents its key (`kDataResource`). The `describedBy` element is used to make reference to `ProgDC` element instances, through a `keyref` definition (`refDescribedByProgram`).

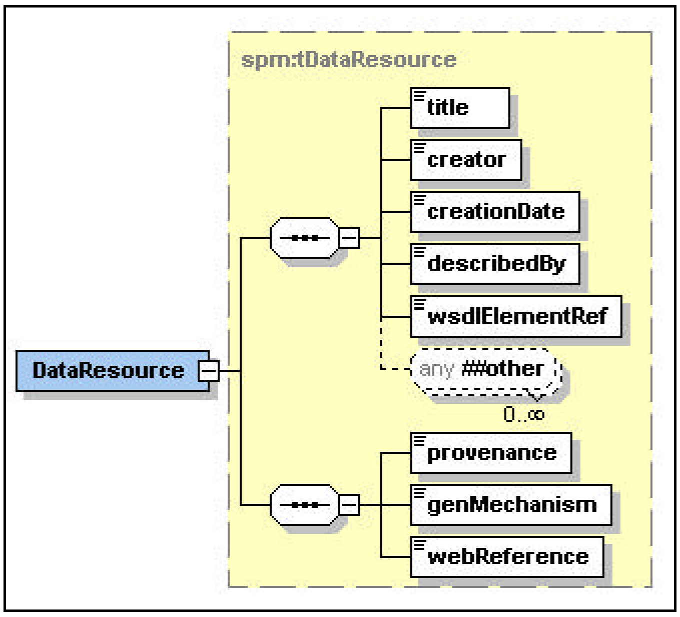


Figure 57: Data Resource XML Schema diagram

```
<xsd:element name="DataResource" type="spm:tDataResource">
     <xsd:key  name="kDataResource">
          <xsd:selector xpath="."/>
          <xsd:field xpath="@idDR"/>
     </xsd:key>
     <xsd:keyref name="refdescribedByProgDC" refer="spm:kProgDC">
          <xsd:selector xpath="."/>
          <xsd:field xpath="spm:describedBy"/>
     </xsd:keyref>
</xsd:element>
```

Figure 58: Data Resource Key and Keyrefs definitions

5.2.1.7 *Workflow*– spm:Workflow

The `workflow` element represents the execution plans to be used by experiments. As shown in Figure 59, a workflow is described by a set of self-explanatory descriptive elements (`title`, `creator` and `creationDate`). In addition, a multiple occurrence of the `wfStep` element determines the workflow composition, which is based on `program` element instances. Finally, the specification of how these steps are logically organized is described in the `specification` element.

As shown in Figure 60, the `workflow` element has a unique attribute that represents its key (`kWorkflow`). The `wfStep` element is used to make reference to `Program` element instances, through a `keyref` definition (`refWfStep`).

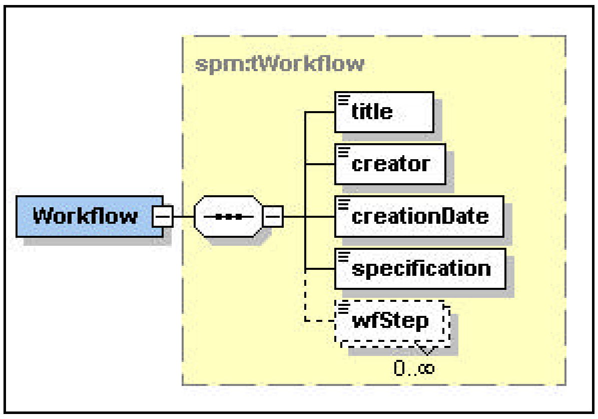


Figure 59: Workflow XML Schema diagram

```
<xsd:element name="Workflow" type="spm:tWorkflow">
    <xsd:key name="kWorkflow">
        <xsd:selector xpath="."/>
        <xsd:field xpath="@idWF"/>
    </xsd:key>
    <xsd:keyref name="refWfStep" refer="spm:kProgram">
        <xsd:selector xpath="."/>
        <xsd:field xpath="spm:wfStep"/>
    </xsd:keyref>
</xsd:element>
```

Figure 60: Workflow Key and Keyrefs definitions

It is worth to mention that SPMW is in its first version, and the workflow description should evolve. For instance, we aim to address abstract workflow definitions in terms of models (instead of just programs), by proposing extensions to the workflow definition language.

5.2.1.8 *Experiment*– spm:Experiment

The `experiment` element represents the use of scientific resources. As shown in Figure 61, it is composed by elements that identify, describe the context of the experiment and capture the history of essays. `Title`, `creator` and `creationDate` are the identification elements. Then, a set of elements contextualize the experiment. The `project` element associates it to a research project. The experiment hypothesis and purpose are stated in the correspondent elements. The `status` element indicates the experiment status, e.g., if it is active, finished, archived, etc. The `report` element is used to capture the scientist final report about the experiment. Finally, the workflow element has multiple occurrences. Each experiment may have one or more workflows associated to it. These workflows are used during the experiment essays.

The many essays of an experiment capture the history of code executions. Each `essay` element is described by the moment it starts (`creationDate` and `creationTime`) and the workflow it is instantiated from (`instanceOf`). The `comment` element captures scientist comments about each essay. The `duration` element is updated as all the executions finish. As the essay is an instance of a workflow, it comprises a set of code executions. Each `execution` element is described by associations to code and data resource elements instances. For each data IO and parameter associated to the code corresponding program interface, a specific data resource is associated. This relationshipis mapped through `dataMatch` and `parmMatch` elements, respectively.

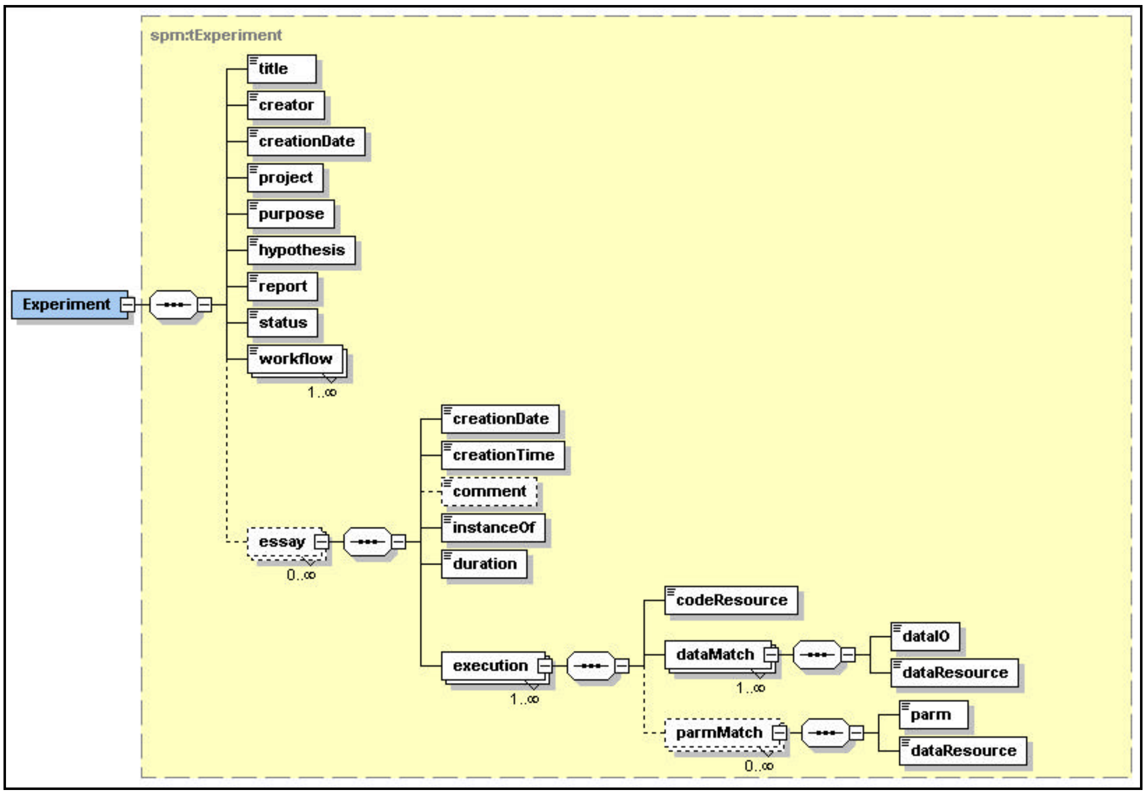


Figure 61: Experiment XML Schema diagram

As shown in Figure 62, the `experiment` element has a unique attribute that represents its key (`kExperiment`). The `workflow` and the `instanceOf` elements are used to make reference to `workflow` element instances, through `keyref` definitions (`refWorkflow` and `refInstanceOf`, respectively). The codeResource and the dataResource elements are used to make reference to code and data resource element instances, through keyref definitions (refCodeResource and refDataResource, respectively).

```
<xsd:element name ="Experiment" type="spm:tExperiment">
    <xsd:key  name="kExperiment">
        <xsd:selector xpath=".”/>
        <xsd:field xpath="@idEx"/>
    </xsd:key>
    <xsd:keyref name="refWorkflow " refer="spm:kWorkflow ">
        <xsd:selector xpath=".”/>
        <xsd:field xpath="spm:workflow "/>
    </xsd:keyref>
    <xsd:keyref name="refInstanceOf" refer="spm:kWorkflow ">
        <xsd:selector xpath="spm:essay"/>
        <xsd:field xpath="spm:instanceOf"/>
    </xsd:keyref>
    <xsd:keyref name="refCodeResource" refer="spm:kCodeResource">
        <xsd:selector xpath="spm:essay/spm:execution"/>
        <xsd:field xpath="spm:codeResource"/>
    </xsd:keyref>
    <xsd:keyref name="refDataResource" refer="spm:kDataResource">
        <xsd:selector xpath="spm:essay/spm:execution/spm:dataMatch"/>
        <xsd:field xpath="spm:dataResource"/>
    </xsd:keyref>
</xsd:element>
```

Figure 62: Experiment Key and Keyrefs definitions

5.2.2 Extending a WSDL document

Taking advantage of the WSDL extensibility element (definitions), the new element may be declared just under it. To illustrate how we extend WSDL with SPMW elements let us take the SPMW program element. In WSDL such element is represented as a port type operation. In SPMW, we have designed a program element which is associated to a WSDL portType operation element instance through the wsdlElementRef element.

Also, the program element makes reference to other SPMW elements, such as data input and output types. Thus, each program element has a set of IOData sub-elements (input, parm and output). Each of these sub-elements should refer to SPMW program data category element instances (ProgDC instances) through the refersTo sub-element, establishing the connection between a program and its inputs and outputs. To associate SPMW ProgDC instances to WSDL types, each ProgDC element refers to the WSDL message part type attribute, through the messagePartType sub-element, while the SPMW program IOData and parameter sub-elements refer to the WSDL message part name attribute.

The WSDL file is duplicated by SRMW and altered to have an extra `import` definition inside it. Through this definition SRMW couples metadata of scientific resources to WSDL elements. Consequently, other applications may have access to all metadata related to some scientific resource published in SRMW as an extended WSDL document. For instance, when queried about a specific program, SRMW would provide an extended WSDL including contents of all related SPMW documents.

6. Using SRMW with Scientific Applications

Among the vast amount of scientific applications, we have had the opportunity to study closely two noteworthy ones: a biocorrosion application and a structural genomic application. A special team in the Research Centre - CENPES-Petrobras deals with the investigation of biocorrosion phenomena. Some of the team specialists worked with us on a joint research project, which allowed us to study a real biocorrosion application. Another group of scientists, working at the Institute of Biophysics Carlos Chagas Filho (IBCCF) deals with structural genomic applications. IBCCF scientists take part on a research project whose main objective is to define a scientific workflow based on bioinformatics programs to build molecular tri-dimensional models. These programs are based on algorithmic models that may generate different and useful implementations. Some IBCCF researchers are working with us on a joint research project where we aim at providing support for scientific workflows and data derivation and provenance. The strong interaction with the scientific specialists has put us close to real problems and requirements.

The main goal of using SRMW with these applications is to analyse the adequacy of SRMW components with respect to supporting the requirements of the applications. The three general requirements are: (i) to handle distribution and heterogeneity, (ii) to describe scientific resources and (iii) to manage scientific workflows and register their use. Those two applications confirm these three requirements. However, they have specific needs and different "weights" on the importance of the three requirements.

In biocorrosion teams, (i) is a basic necessity since the team is highly heterogeneous, that is, the scientists come from different application domains, and work with specific models in sub-teams physically distributed (not necessarily through the Web, but through some Intranet). Usually, they exchange data analysis, but they do not share programs or models between sub-teams. However, sharing programs and models would increase the productivity among scientists from different sub-teams. Their main problem relates to finding the right model for each case, and correctly executing the correspondent program. Therefore, describing programs and grouping models (item (ii)) are a crucial requirement in biocorrosion teams. Since models are often reused, once they are published in SRMW, browsing model's descriptions and model's usage facilitates biocorrosion scientists. This means browsing abstract workflows and experiments. Finally, building scientific workflows is not a strong requirement among biocorrosion scientists. Thus, item (iii) is necessary but not vital as the other items. However, we believe that with SRMW support, scientific workflows will naturally become a requirement, even to biocorrosion scientists. In summary, through biocorrosion teams we were able to analyse metamodel issues with respect to scientific models and programs, as well as tools to navigate through published resources.

In biophysics teams, (i) is also a basic necessity but for different reasons. Biophysics are usually part of homogeneous teams. However, they are usually organized in sub-teams involved in developing or using a specific program. These sub-teams are physically distributed and exchange their programs throughout the Web. Typically these programs are available at Web sites for online usage, where associated user guides help the user in filling up parameters and program inputs. In this case, item (ii) is important but not critical. Describing programs is useful, but grouping programs according to their models is not as important as it is the case of biocorrosion teams. Finally, item (iii) is crucial to biophysics teams. Their main difficulty relates to the combination of a program output with another program input, meaning they need to execute scientific workflows, composed by heterogeneous and distributed programs. Web services are a right solution as provides dynamic interaction between these programs, facilitating their combination and execution as workflows. Besides, monitoring these executions is even more important, as biophysics, as well as many other scientists, need to keep track of data provenance. Once published in SRMW, biophysics workflows are easily monitored and experiments are registered into its metadata repository. In summary, through biophysics teams we were able to analyse metamodel issues with respect scientific workflows and experiments, as well as tools to support workflow definition, instantiation and execution.

In SRMW the scientific resources that are useful to an application should be available at first. For example, scientific models should be available to compose workflows, upon which an experiment is published. In this direction, this section describes how scientists have published some of their resources and how this is useful to perform new experiments.

The next sub-sections show SRMW being used within a souring application (section 6.1) and a structural genomic application (section 6.2), presenting their publications through the SRMW Publication module and the SPMW metamodel elements.

6.1 Souring Application

Corrosion monitoring on oil platforms over the Brazilian coastal zone is one of the main concerns of scientists from CENPES-Petrobrás. The production of oil in deep water reservoirs benefits from the injection of sea water into the reservoir. The process of water injection increases the pressure inside the reservoir, facilitating the oil exploitation. Before being injected, the sea water is treated with chloride, filtrated and deoxygenated.

A typical biocorrosion application is the investigation of the causes of oil pipe obstructions. There are different hypotheses to consider. The obstruction may be caused by micro-organism activity (souring), by sand accumulation or by corrosion products accumulation. For each hypothesis a new experiment is initiated.

Souring is one of the main problems that may occur during an oil reservoir lifetime. This problem is believed to be caused by sulfate-reducing bacteria (SRB), present in the reservoir. These bacteria are strict anaerobic micro-organisms that accumulate themselves like biofilms on oil reservoir porous walls, and reduce sulfate that come with the water to hydrogen sulfite. The problem becomes worse when the sea water is not treated adequately before injection. In these cases, solids characterized by bacteria colonies besides inorganic compounds can be carried into the oil reservoir. Therefore, the quality of the water introduced in the pipes and the quality of the reservoir internal cover are both factors that may bring souring problems to the oil extraction process.

To illustrate the use of the SPMW metamodel we have published some of the useful resources to a specific souring application called Cabiunas case study. The Cabiunas case study has started based on the identification of the appearance of the deadly hydrogen sulphide gas (H_2S) in an oil-water separation storage area in the city of Cabiunas, RJ.

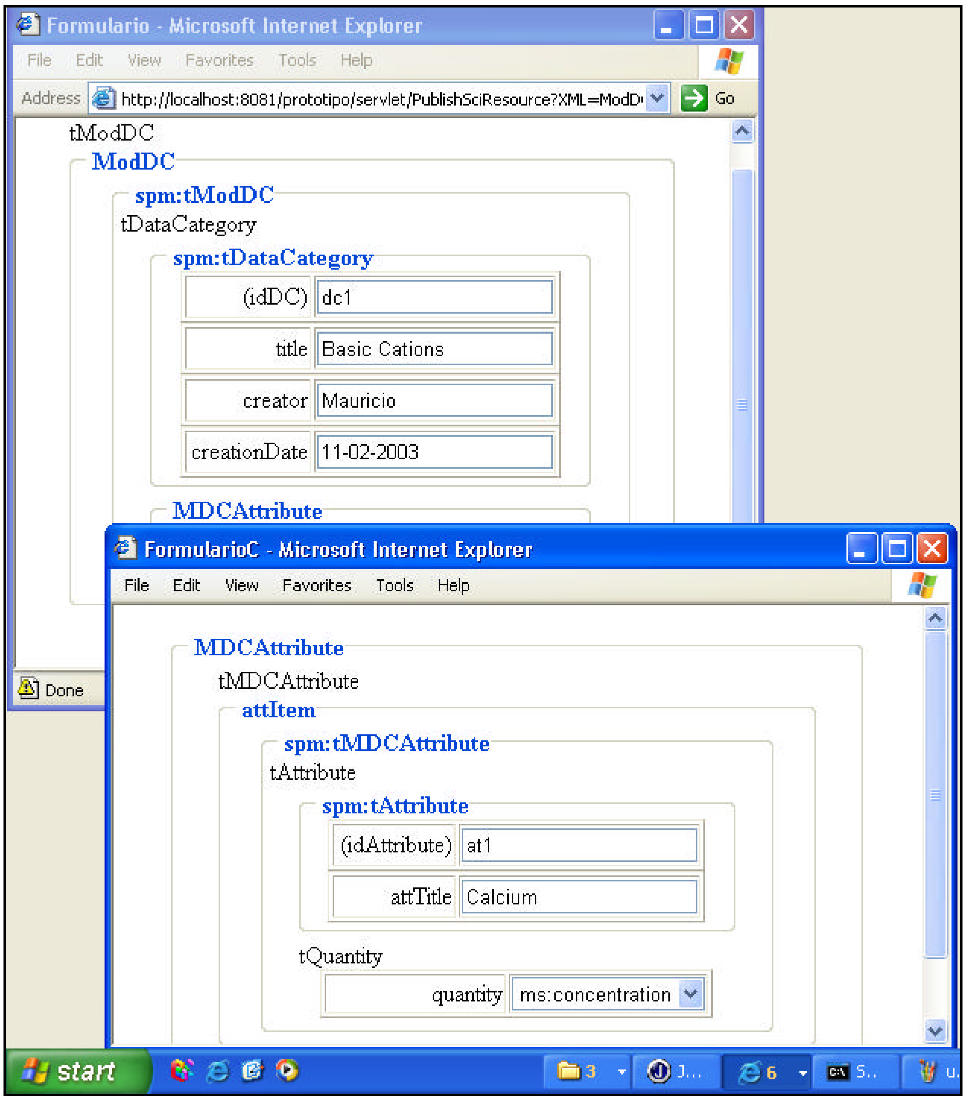
We start by publishing the Kuznetsova model, and the corresponding data categories (section 6.1.1), then we publish the Kuznetsova program and its data categories (section 6.1.2). In section 6.1.3 we publish an available code resource for the Kuznetsova program, while in section 6.1.5, we publish an available data resource for the Cabiunas case study. In section 6.1.4 we publish the Kuznetsova workflow and associate it to the experiment published in section 6.1.7. Finally, in 6.1.6 we show the user browsing resources to perform a new experiment, which is illustrated in section 6.1.8.

6.1.1 Publishing Kuznetsova model

We have started by publishing a very simple mathematical model known as Kuznetsova model (KUZNETSOVA, V. A., 1960). To publish the Kuznetsova model, the publisher should first publish the model data categories used as input and output for that model. The *input model data category* for the Kuznetsova model involves information extracted from the chemical analysis of a water sample. The Kuznetsova model needs the concentration values of four chemical elements (also known as basic cations), which may be described as a composite of *model attributes*: Calcium, Magnesium, Potassium, and Sodium. Figure 63 shows how the scientist uses the SRMW Publication module interface to describe the Model Data Category for the Basic Cations input. This module is responsible for translating the information into the XML document shown in Figure 64, which is valid according to the SPMW XML Schema.

The Kuznetsova model is based on a formula that relates the concentration values of these four chemical elements, and generates a non-dimensional number. Based on this result and on the Chloride concentration value, the Kuznetsova model informs if there is the ideal condition for sulfate-reduction bacteria growth, and if so, it returns a graphic curve of the hydrogen sulfide gas (H_2S) production tendency, observed in time. Thus, this output may be described as an *output model data category* for the Kuznetsova model, with *model attributes* representing the graphic axes (x, y) information. One attribute corresponds to time (x-axis), and the other attribute corresponds to hydrogen sulfide production (y-axis).

There are different curves to consider according to the Chloride concentration value, while the storage time determines in which point of the curve is the bacteria activity at the moment. Chloride concentration and storage time may be described as *model parameters* to the Kuznetsova model, as these values guide the plotting of the right gas production graphic.



Formulario - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8081/prototipo/servlet/PublishSciResource?XML=ModD
Go
Done
FormularioC - Microsoft Internet Explorer
start

tModDC
ModDC
spm:tModDC
tDataCategory
spm:tDataCategory

(idDC)	dc1
title	Basic Cations
creator	Mauricio
creationDate	11-02-2003

MDCAttribute

MDCAttribute
tMDCAttribute
attItem
spm:tMDCAttribute
tAttribute
spm:tAttribute

(idAttribute)	at1
attTitle	Calcium

tQuantity

quantity	ms:concentration

Figure 63: Model Data Category Basic Cations Publication

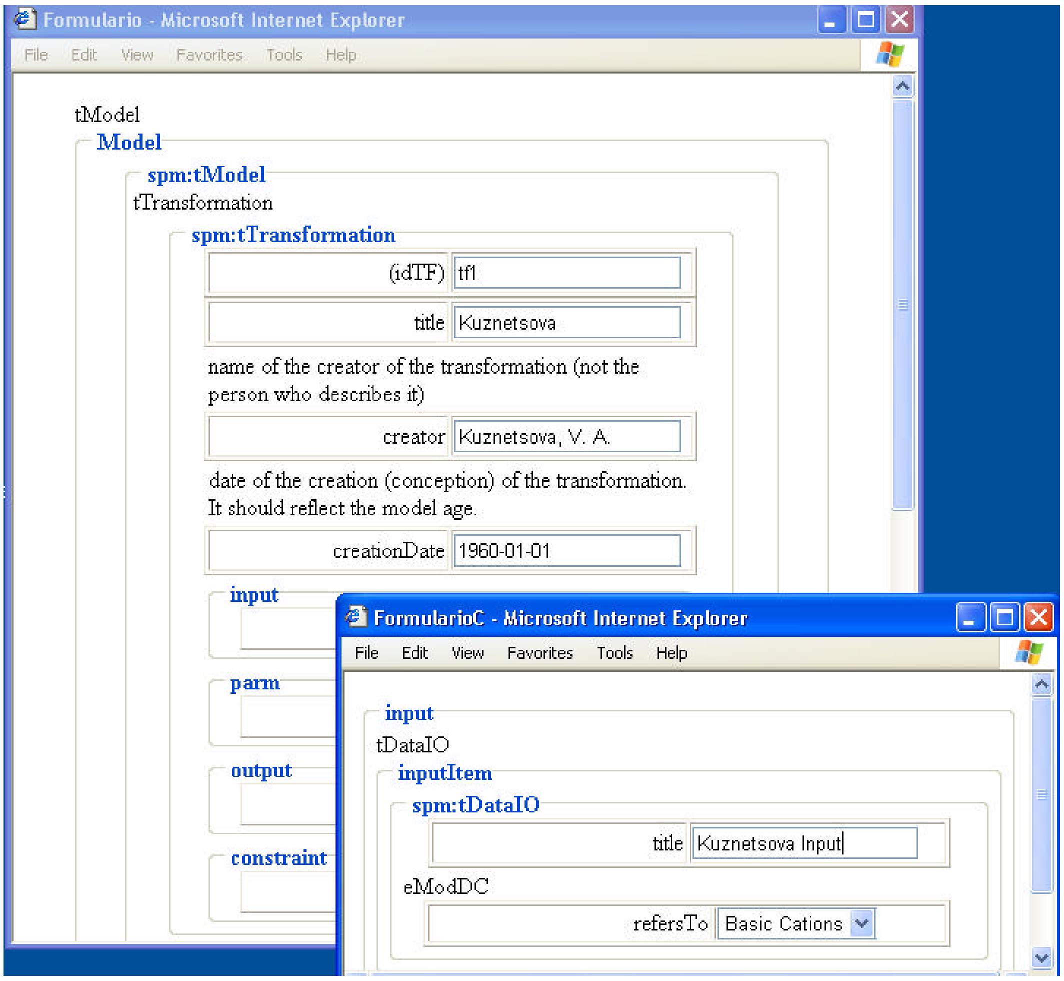
```
<ScientificResourceDefinitions>
    <ModDC idDC="dc1">
        <title>Basic Cations</title>
        <creator>Mauricio</creator>
        <creationDate>2003-02-11</creationDate>
        <MDCAttribute>
            <attItem idAttribute="at1">
                <attTitle>Calcium</attTitle>
                <quantity>ms:concentration</quantity>
            </attItem>
        </MDCAttribute>
        <MDCAttribute>
            <attItem idAttribute="at2">
                <attTitle>Magnesium</attTitle>
                <quantity>ms:concentration</quantity>
            </attItem>
        </MDCAttribute>
        <MDCAttribute>
```

```
                <attItem idAttribute=”at3”>
                    <attTitle>Potassium</attTitle>
                    <quantity>ms:concentration</quantity>
                </attItem>
            </MDCAttribute>
            <MDCAttribute>
                <attItem idAttribute=”at4”>
                    <attTitle>Sodium</attTitle>
                    <quantity>ms:concentration</quantity>
                </attItem>
            </MDCAttribute>
        </ModDC>
    </ScientificResourceDefinitions>
```

Figure 64: Model Data Category Basic Cations XML document

After publishing the related data categories, the publisher may describe the Kuznetsova model, and relate it to two *I/O data* and one *parm*, which refer to *input, output and parm model data categories*, already published. The Model description involves information about its scope, area, purpose, hypothesis, etc. Figure 65 and Figure 66 show how the publisher can describe the Kuznetsova model using the Publication module interface.

The Kuznetsova model applies only to environments where there might be bacteria activity. Thus, a *constraint* should be defined over the Chloride parameter, meaning it should assume positive values below 140.000 ppm, as Figure 66 shows. The whole model description generates the XML document presented in Figure 67.



Formulario - Microsoft Internet Explorer
File Edit View Favorites Tools Help

tModel

Model

spm:tModel

tTransformation

spm:tTransformation

(idTF) tf1

title Kuznetsova

name of the creator of the transformation (not the person who describes it)

creator Kuznetsova, V. A.

date of the creation (conception) of the transformation. It should reflect the model age.

creationDate 1960-01-01

input

parm

output

constraint

FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help

input

tDataIO

inputItem

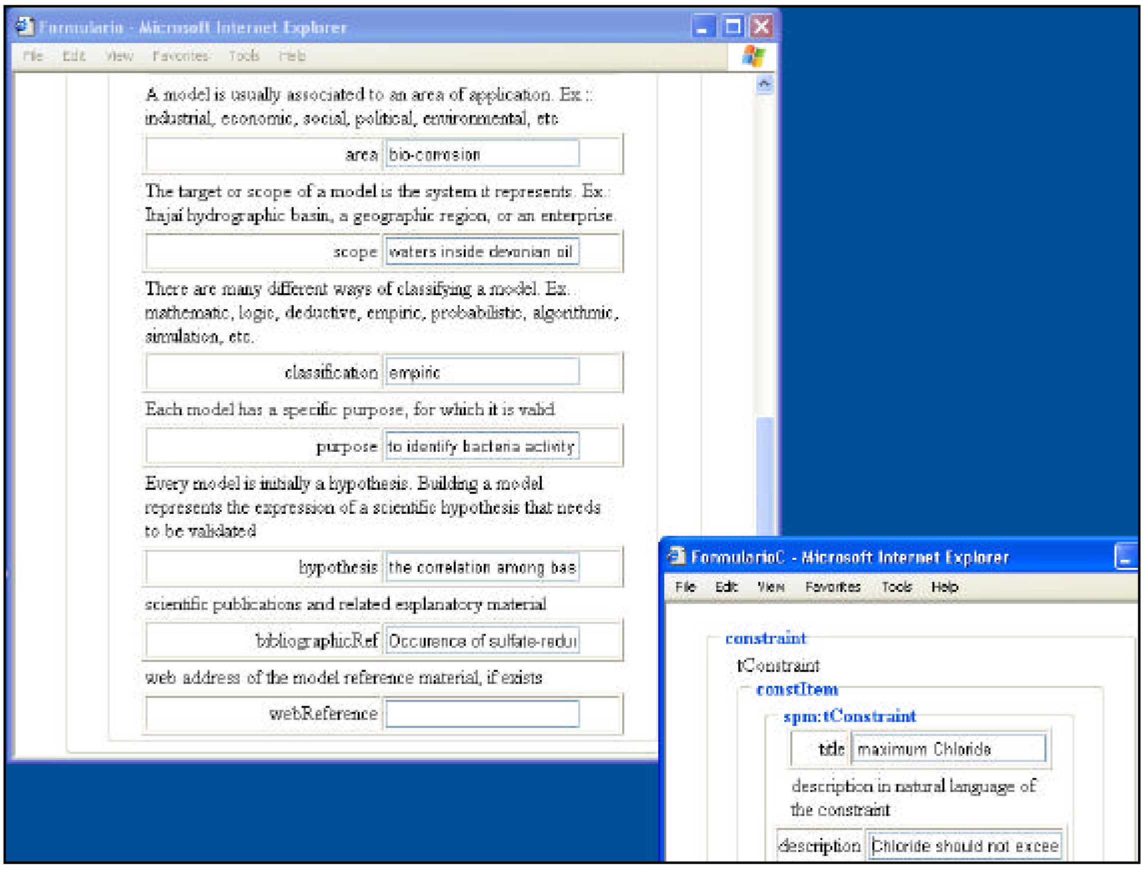
spm:tDataIO

title Kuznetsova Input

eModDC

refersTo Basic Cations

Figure 65: Model Publication for Kuznetsova Model (part I)



Formulario - Microsoft Internet Explorer
File Edit View Favorites Tools Help

A model is usually associated to an area of application. Ex.: industrial, economic, social, political, environmental, etc

area: bio-corrosion

The target or scope of a model is the system it represents. Ex.: Itajaí hydrographic basin, a geographic region, or an enterprise.

scope: waters inside devonian oil

There are many different ways of classifying a model. Ex. mathematic, logic, deductive, empiric, probabilistic, algorithmic, simulation, etc.

classification: empiric

Each model has a specific purpose, for which it is valid

purpose: to identify bacteria activity

Every model is initially a hypothesis. Building a model represents the expression of a scientific hypothesis that needs to be validated

hypothesis: the correlation among bas

scientific publications and related explanatory material

bibliographicRef: Occurence of sulfate-redu

web address of the model reference material, if exists

webReference:

FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help

constraint

tConstraint

constItem

spm:tConstraint

title: maximum Chloride

description in natural language of the constraint

description: Chloride should not excee

Figure 66: Model Publication for Kuznetsova Model (part II)

```
<ScientificResourceDefinitions>
    <Model idTF="tf1">
        <title>Kuznetsova</title>
        <creator>Kuznetsova, V. A.</creator>
        <creationDate>1960-01-01</creationDate>
        <input>
            <inputItem>
                <title>Kuznetsova Input</title><refersTo>dc1</refersTo>
            </inputItem>
        </input>
        <parm>
            <parmItem>
               <title>Kuznetsova configuration</title><refersTo>dc2</refersTo>
            </inputItem>
        </parm>
        <output>
            <outputItem>
               <title>Kuznetsova Output</title><refersTo>dc3</refersTo>
            </outputItem>
        </output>
        <area>bio-corrosion</area>
        <scope>waters inside devonian oil reservoirs</scope>
        <classification>empiric </classification>
        <purpose>to identify bacteria activity in oil reservoirs</purpose>
        <hypothesis>The correlation among basic cations indicates
          a proper environment for bacteria activity</hypothesis>
        <bibliographicRef>Ocurrence of sulfate-reducing organisms in oil-bearing
          formations of Kuibyshev region with reference to salt composition of layer
          waters. Mikrobiologiya, 29 pp. 408-414, 1960. </bibliographicRef>
    </Model>
</ScientificResourceDefinitions>
```

Figure 67: Model XML document

6.1.2 Publishing Kuznetsova Program

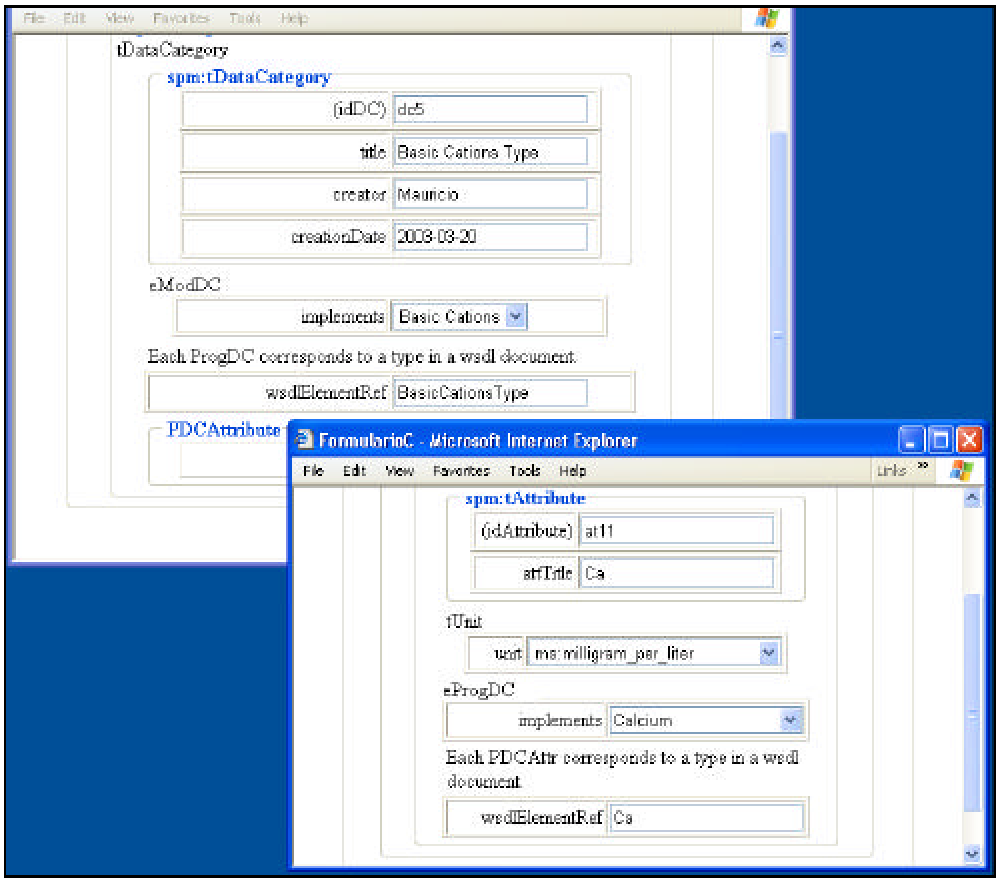
Now, let us consider that a scientist has implemented a *program* based on the Kuznetsova model and wants to make it available for the team. First of all, a Web service for that program should be built. Then, while publishing the Kuznetsova program in SRMW, the WSDL file location is informed and extended through SPMW elements. Figure 68 shows the WSDL file for the Kuznetsova program. Based on some information in this file, the SRMW Publication module builds its publication interface to capture the extended program description.

```
<definitions>
    <types>
        <xsd:schema ... >
            <xsd:complexType name="ConfigurationType">
                <xsd:sequence>
                    <xsd:element name="chloride" type="xsd:float"/>
                    <xsd:element name="storageTime" type="xsd:integer"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:simpleType name="cMaxChloride">
                <xsd:restriction>
                    <xsd:maxExclusive value="140000"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:complexType name="BasicCationsType">
                <xsd:sequence>
                    <xsd:element name="Ca" type="xsd:float"/>
                    <xsd:element name="Mg" type="xsd:float"/>
                    <xsd:element name="K" type="xsd:float"/>
                    <xsd:element name="Na" type="xsd:float"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="H2SproductionType">
                <xsd:sequence>
                    <xsd:element name="time" type="xsd:float"/>
                    <xsd:element name="H2Sproduction" type="xsd:float"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </types>
    <message name="codeInputmsg">
        <part name="configurationPart" type="ConfigurationType"/>
        <part name="BasicCationsPart" type="BasicCationsType"/>
    </message>
    <message name="codeOutputmsg">
        <part name="H2SproductionPart" type="H2SproductionType"/>
    </message>
    <portType name="kuznetsovaPortType">
        <operation name="kuznetsovaOperation">
            <input message="tns:codeInputmsg"/>
            <output message="tns:codeOutputmsg"/>
        </operation>
    </portType>
...
</definitions>
```

Figure 68: WSDL document for Kuznetsova Program (abstract part)

Similarly to the model publication, the scientist should first describe the related program data categories. The *program input data category* for "basic cations" includes a *program attribute* for each of the four elements described as *model attributes*. In the case of the Kuznetsova program input data category, each *program attribute* describes the *model attribute*, described to be a concentration value, as a "float" type containing values in "mg/l" unit. WSDL already describes these attributes and their primitive types, while SPMW is in charge of describing the unit used and associating it to its correspondent quantity. Figure 69 shows the publication of Kuznetsova program and how its description refers to WSDL elements, seen in Figure 68.

A *program output data category* should correspond to the *model output data category* "H_2S production graphic". The x-axis corresponds to a *program attribute* that implements the *model attribute* for *time*, and its unit may be, for instance, "day". The y-axis corresponds to another *program attribute*, which implements the *model attribute* for "hydrogen sulfide gas production", having its unit as "mg/l".


File Edit View Favorites Tools Help
FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links

tDataCategory
spm:tDataCategory

(idDC)	dc5
title	Basic Cations Type
creator	Mauricio
creationDate	2003-03-20

eModDC

implements	Basic Cations

Each ProgDC corresponds to a type in a wsdl document

wsdlElementRef	BasicCationsType

PDCAttribute

spm:tAttribute

(idAttribute)	at11
attTitle	Ca

tUnit

unit	ms:milligram_per_liter

eProgDC

implements	Calcium

Each PDCAttr corresponds to a type in a wsdl document

wsdlElementRef	Ca

Figure 69: Kuznetsova Program Publication

After publishing both program data categories, the publisher may describe the Kuznetsova *program*, and associate it to two *I/O data* and one *parm*, which refer to the *input*, *output* and *parm program data categories*, previously published. Also, a constraint on the Chloride parameter is described based on the XML schema restriction element defined in the WSDL document. Finally, the publisher may associate the *program* to the *model* it implements. The XML document for the Basic Cations Program Data Category and the Kuznetsova Program are presented in Figure 70.

```
<ScientificResourceDefinitions>
    <ProgDC idDC="dc11">
        <title>Basic Cations Type</title>
        <creator>Mauricio</creator>
        <creationDate>2003-03-20</creationDate>
        <wsdlElementRef>BasicCationsType</wsdlElementRef>
        <PDCAttribute>
            <attItem idAttribute="">
                <attTitle>Ca</attTitle>
                <unit>ms:milligram_per_liter</unit>
                <wsdlElementRef>Ca</wsdlElementRef>
            </attItem>
        </PDCAttribute>
    </ProgDC>
</ScientificResourceDefinitions>
<ScientificResourceDefinitions>
    <Program idTF="tf2">
        <title>Kuznetsova</title>
        <creator>Mauricio</creator>
        <creationDate>2003-03-20</creationDate>
        <input>
            <inputItem>
                <title>Kuznetsova input</title>
                <refersTo>dc11</refersTo>
            </inputItem>
        </input>
        <parm>
            <parmItem>
                <title>Kuznetsova Configuration</title>
                <refersTo>dc12</refersTo>
            </parmItem>
        </parm>
        <output>
            <outputItem>
                <title>Kuznetsova output</title>
                <refersTo>dc13</refersTo>
            </outputItem>
        </output>
        <constraint>
            <constItem>
                <title>cMaxChloride</title>
                <description>Chloride should not exceed 140.000 ppm</description>
                <expression>maxExclusive value="140000"</expression>
            </constItem>
        </constraint>
        <implementationLanguage>C++</implementationLanguage>
        <version>1.0</version>
        <implements>tf1</ implements>
        <wsdlElementRef>KuznetsovaOperation</wsdlElementRef>
    </Program>
</ScientificResourceDefinitions>
```

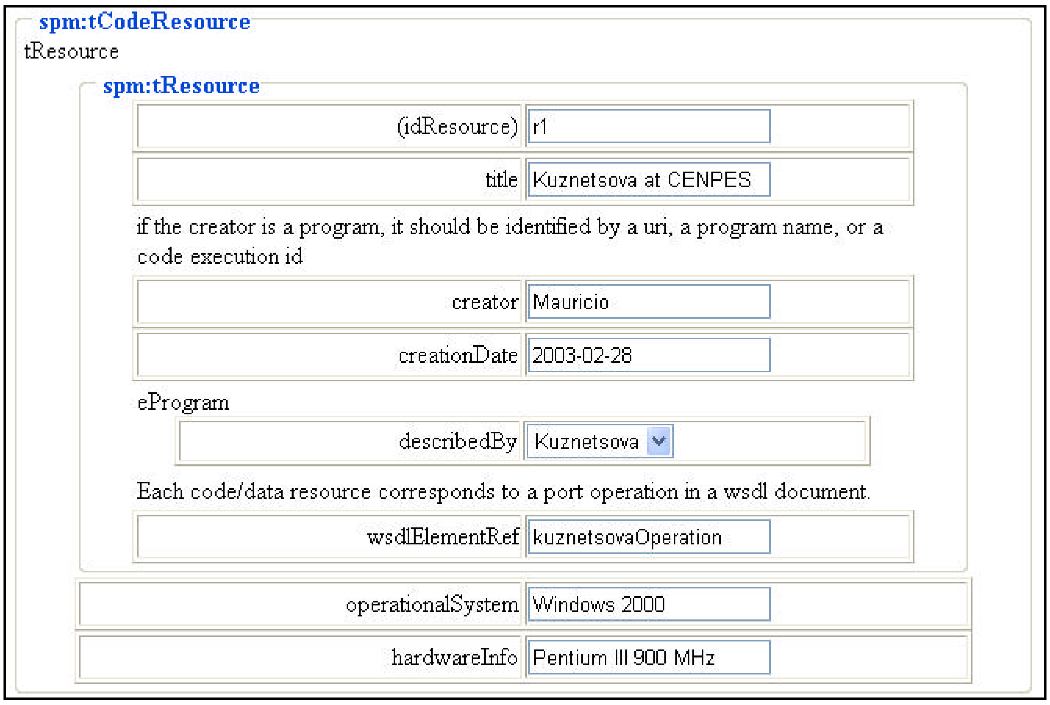
Figure 70: Program and ProgDC XML document for Kuznetsova Program

6.1.3 Publishing Kuznetsova Code Resource

After publishing the Kuznetsova program interface it is now possible to publish a related code resource. This compiled code is available for execution as a Web service, and its concrete description is presented in Figure 71. Similarly to the Program publication, SPMW provides description elements for extending the WSDL contents. The SRMW Publication module interface, presented in Figure 72, shows how to associate the code resource to its correspondent WSDL Port Type Operation and SPMW Program elements.

```
<definitions>
...
    <binding name="kuznetsovaBinding" type="tns:kuznetsovaPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="kuznetsovaOperation">
            <soap:operation
soapAction="capeconnect:kuznetsova:kuznetsovaPortType#kuznetsovaOperation"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.your-company.com/kuznetsova/binding" use="encoded"/>
            </input>
            <output>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.your-company.com/kuznetsova/binding" use="encoded"/>
            </output>
        </operation>
    </binding>
    <service name="kuznetsova">
        <port binding="tns:kuznetsovaBinding" name="kuznetsovaPort">
            <soap:address location="http://localhost:9000/ccx/kuznetsova"/>
        </port>
    </service>
</definitions>
```

Figure 71: WSDL document for Kuznetsova Program (concrete part)



spm:tCodeResource

tResource

spm:tResource

(idResource)	r1
title	Kuznetsova at CENPES

if the creator is a program, it should be identified by a uri, a program name, or a code execution id

creator	Mauricio
creationDate	2003-02-28

eProgram

describedBy	Kuznetsova

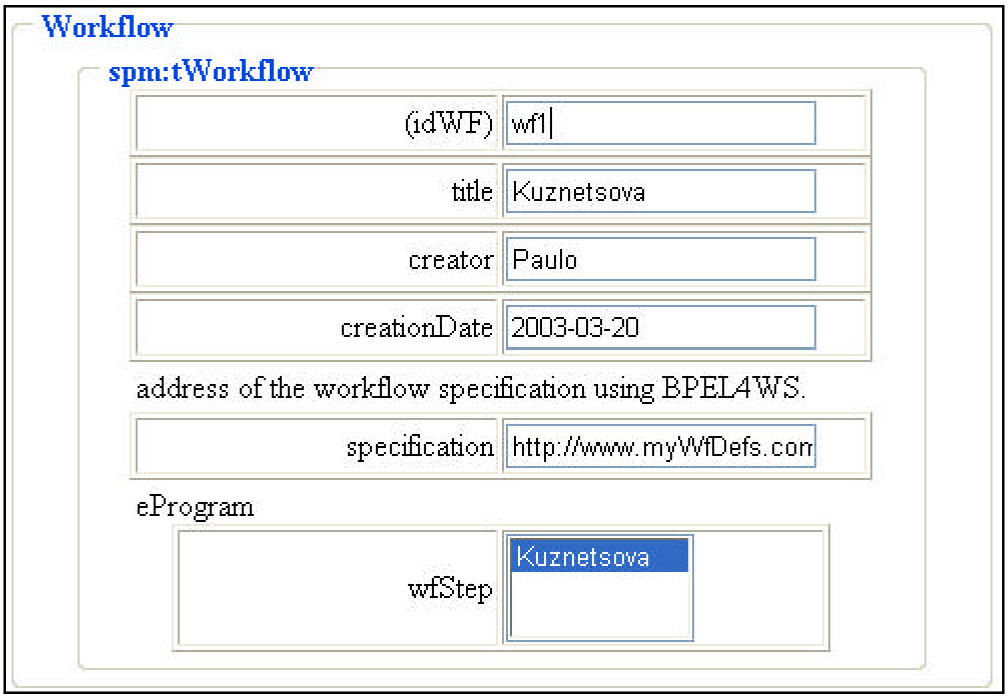
Each code/data resource corresponds to a port operation in a wsdl document.

wsdlElementRef	kuznetsovaOperation
operationalSystem	Windows 2000
hardwareInfo	Pentium III 900 MHz

Figure 72: Kuznetsova Code Resource Publication

6.1.4 Publishing the Kuznetsova Workflow

To make the Kuznetsova program available to experiments, we need to declare it as part of a workflow. For our example, a one step workflow is sufficient. A Workflow edition tool helps on the specification of the Kuznetsova workflow, which is then informed to the Publication module, as shown in Figure 73. Then the Publication model stores the workflow description as the XML document presented in Figure 74.


Workflow

spm:tWorkflow

(idWF)	wf1
title	Kuznetsova
creator	Paulo
creationDate	2003-03-20

address of the workflow specification using BPEL4WS.

specification	http://www.myWfDefs.com

eProgram

wfStep	Kuznetsova

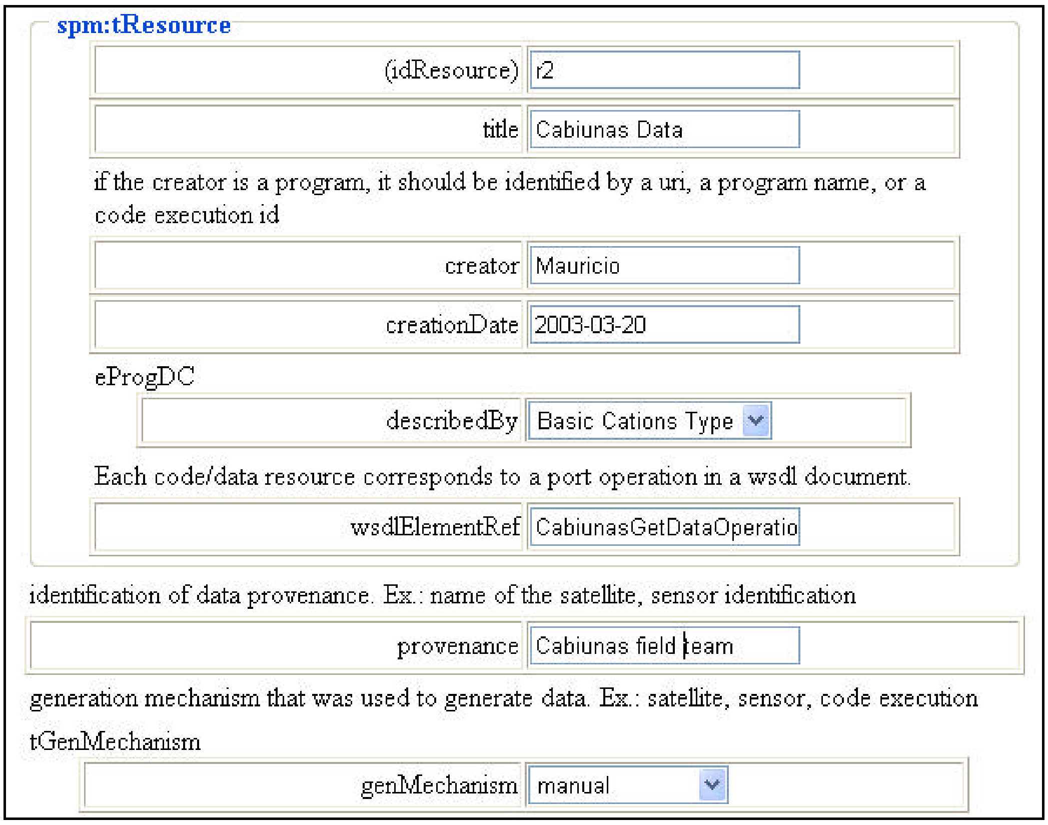
Figure 73: Kuznetsova Workflow Publication

```
<ScientificResourceDefinitions>
<Workflow idWF="wf1">
      <title>Kuznetsova</title>
      <creator>Paulo</creator>
      <creationDate>2003-03-21</creationDate>
      <specification>http://www.myWfDefs.br/kuznetsova.bpel</specification>
</Workflow>
</ScientificResourceDefinitions>
```

Figure 74: Kuznetsova Workflow XML document

6.1.5 Publishing Cabiunas Data

During Cabiunas case study, a field team had manually collected some samples of the water inside Cabiunas tanks. To have these data available as a scientific resource in SRMW, the scientist needs to publish them as a Web service. Also, while publishing these data, to use them as input to the Kuznetsova program requires that their type definition be compatible to the type definition referred by the Kuznetsova program input, i.e., the Basic Cations data category. Figure 75 shows how the Cabiunas data resource is described, and how it is associated to its correspondent Program data category and WSDL Port Type Operation elements.



spm:tResource

(idResource)	r2
title	Cabiunas Data

if the creator is a program, it should be identified by a uri, a program name, or a code execution id

creator	Mauricio
creationDate	2003-03-20

eProgDC

describedBy	Basic Cations Type

Each code/data resource corresponds to a port operation in a wsdl document.

wsdlElementRef	CabiunasGetDataOperatio

identification of data provenance. Ex.: name of the satellite, sensor identification

provenance	Cabiunas field team

generation mechanism that was used to generate data. Ex.: satellite, sensor, code execution

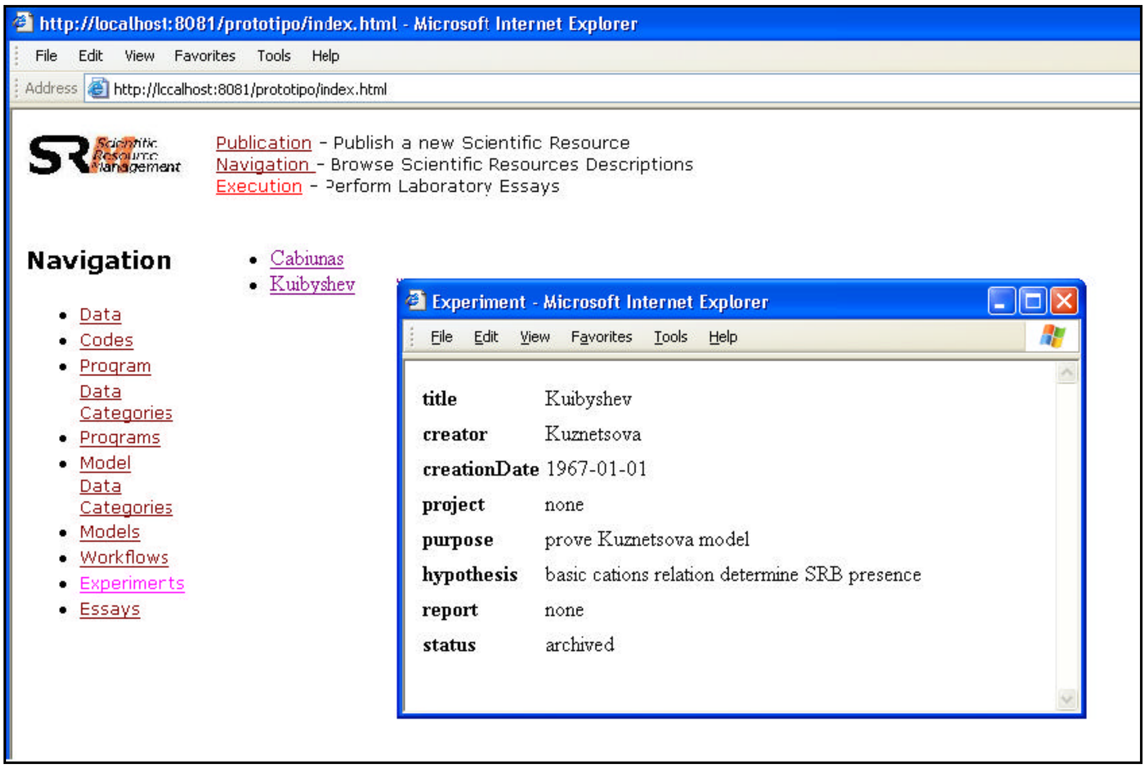
tGenMechanism

genMechanism	manual

Figure 75: Cabiunas Data Resource Publication

6.1.6 Navigating through Biocorrosion Resources

Let us suppose, we had also published the Kuibyshev experiment, which was reported in the Kuznetsova model paper. Figure 76 shows how the SRMW Navigation module helps a scientist on finding the scientific resources that could help a new case study. After browsing through the published experiments, the scientist has access to the details of Kuibyshev experiment. The scientist realizes this experiment used the Kuznetsova model, which applies to oil reservoirs (scope). However, when analysing model inputs and parameters, the specialist concludes the model could also be useful to oil tanks. Therefore, the specialist chooses to run the Kuznetsova program on Cabiunas sample data.


http://localhost:8081/prototipo/index.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8081/prototipo/index.html
SRM Scientific Resource Management
Experiment - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Publication - Publish a new Scientific Resource
Navigation - Browse Scientific Resources Descriptions
Execution - Perform Laboratory Essays

Navigation

- Data
- Codes
- Program Data Categories
- Programs
- Model Data Categories
- Models
- Workflows
- Experiments
- Essays

- Cabiunas
- Kuibyshev

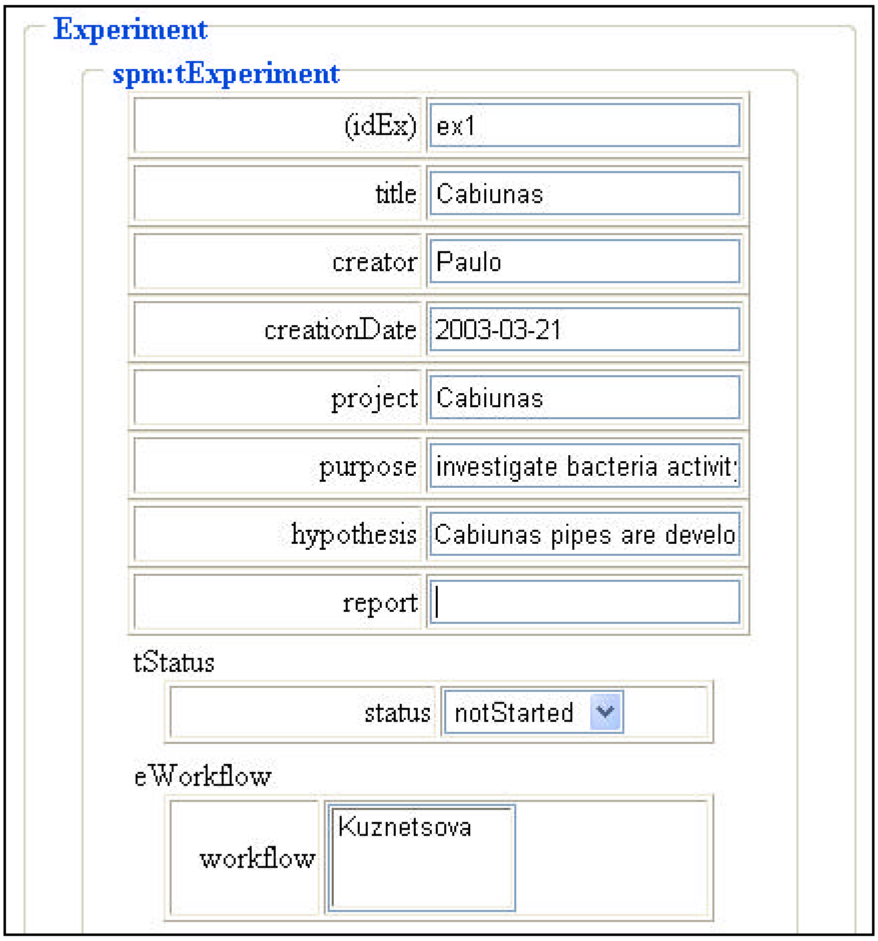
title	Kuibyshev
creator	Kuznetsova
creationDate	1967-01-01
project	none
purpose	prove Kuznetsova model
hypothesis	basic cations relation determine SRB presence
report	none
status	archived

Figure 76: Navigating through Experiments

It is worth to notice the importance of having a metamodel that represents explicitly models and programs. The analysis of models used in previous experiments allowed the scientist to choose the Kuznetsova program.

6.1.7 Publishing the Cabiunas Experiment

To start a new experiment in SRMW the scientist first publishes it. Figure 77 shows how to describe the Cabiunas experiment, and how to associate it to the workflows it is supposed to be using during essays. Both Workflow and Experiment XML documents are presented in Figure 78. Note that the experiment document does not show any essays, meaning it has not started yet.



(idEx)	ex1
title	Cabiunas
creator	Paulo
creationDate	2003-03-21
project	Cabiunas
purpose	investigate bacteria activit
hypothesis	Cabiunas pipes are develo
report	

tStatus

status	notStarted

eWorkflow

workflow	Kuznetsova

Figure 77: Cabiunas Experiment Publication

```
<ScientificResourceDefinitions>
<Experiment idEx ="ex1">
      <title>Cabiunas</title>
      <creator>Paulo</creator>
      <creationDate>2003-03-21</creationDate>
      <project>Cabiunas</project>
      <purpose>Investigate bacteria activity on oil-water storage separation
tanks</purpose>
      <hypothesis>Cabiunas pipes are developing bacteria activity</hypothesis>
      <report></report>
      <status>notStarted</status>
      <workflow >wf1</workflow >
</Experiment>
</ScientificResourceDefinitions>
```

Figure 78: Cabiunas Experiment XML document

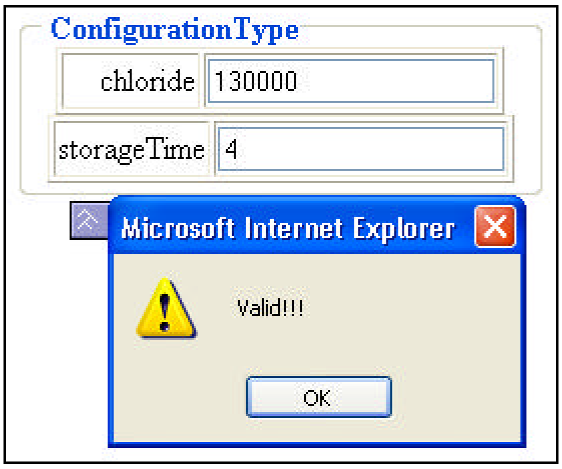
6.1.8 Executing the Cabiunas Experiment

After publishing an experiment, the specialist may then start it, by providing all program input data and parameter values. Considering that parameter values, in Cabiunas case study, were not previously published as a data resource, SRMW helps the user on filling it up on the fly, as shown in Figure 79. The constraint defined on the

Chloride parameter allows a dynamic validation, guiding the user on filling up valid values.

In addition, the Experimentation module helps the user on choosing the adequate input data, considering only those data resources that are described by the same Kuznetsova program input data category. In this case, the Cabiunas sample data will be an option. Finally, the scientist is now able to "execute" the experiment.

Once the experiment results confirm the presence of the sulfate-reduction bacteria, the specialist decides to finish the Cabiunas experiment. During the experiment, the Experimentation module automatically updates the Cabiunas experiment document, registering all the related essays. Then, the user may choose to finish the experiment, and completes the diagnosis report on Cabiunas experiment. The Cabiunas experiment will now be available for other scientists, who will find it through the Navigation module.


ConfigurationType

chloride 130000

storageTime 4

Microsoft Internet Explorer

Valid!!!

Figure 79: Kuznetsova parameter input

6.2 Structural Genomic Application

In the last decades, biochemical laboratories started to perform *in silico* scientific experiments, along with the traditional *in vitro*. Structural genomic applications are typically used to perform *in silico* experiments. As introduced in section 2.2, an IBCCF application called MHOLline (RÖSSLE, S., RIBEIRO, S., *et al.*, 2002), uses the comparative modelling approach for the prediction of protein three-dimensional structures. Nowadays, IBCCF coordinates the Rio de Janeiro Bioinformatics Laboratory and thus receives many requests of structural prediction, which use to be handled one at a time by selected bioinformatics specialists. The MHOLline idea is to enable large-scale modelling by assembling programs on an automated workflow. This would allow biologists to reach structural prediction without depending upon bioinformatics specialists.

MHOLline combines a specific set of programs for the comparative modelling approach (presented in Figure 4). For template structure identification it uses the BLAST algorithm searches (NCBI, 2002). A refinement in the template search step was implemented by a program called BATS (Blast Automatic Targeting for Structures) (RÖSSLE, S., RIBEIRO, S., *et al.*, 2002), where template target sequences are selected from the BLAST output file depending on the given scores for expectation values, identity and sequence coverage. Automated alignment and model building is carried out by a third program, MODELLER (SALI, A., 2001), and models are evaluated using PROCHECK (LASKOWSKI, R. A., MACARTHUR, M. W., MOSS, D. S., THORNTON, J. M., 1993) program scores.

To illustrate the use of the SPMW metamodel we have published useful resources to build and execute the MHOLline workflow. In the next subsections we describe the publication of some of these resources.

We start by publishing the BLAST algorithm, and the corresponding data categories (section 6.2.1), then we publish the BLASTP program and its data categories (section 6.2.2). In section 6.2.3 we publish an available code resource for the BLASTP program, while in section 6.2.5, we publish an available data resource for the Genoma experiment. In section 6.2.4 we publish the MHOLline workflow and associate it to the experiment published in section 6.2.6. Finally, in 6.2.7 we show the user browsing resources to perform a new experiment, which is illustrated in section 6.2.8.

6.2.1 Publishing BLAST algorithm

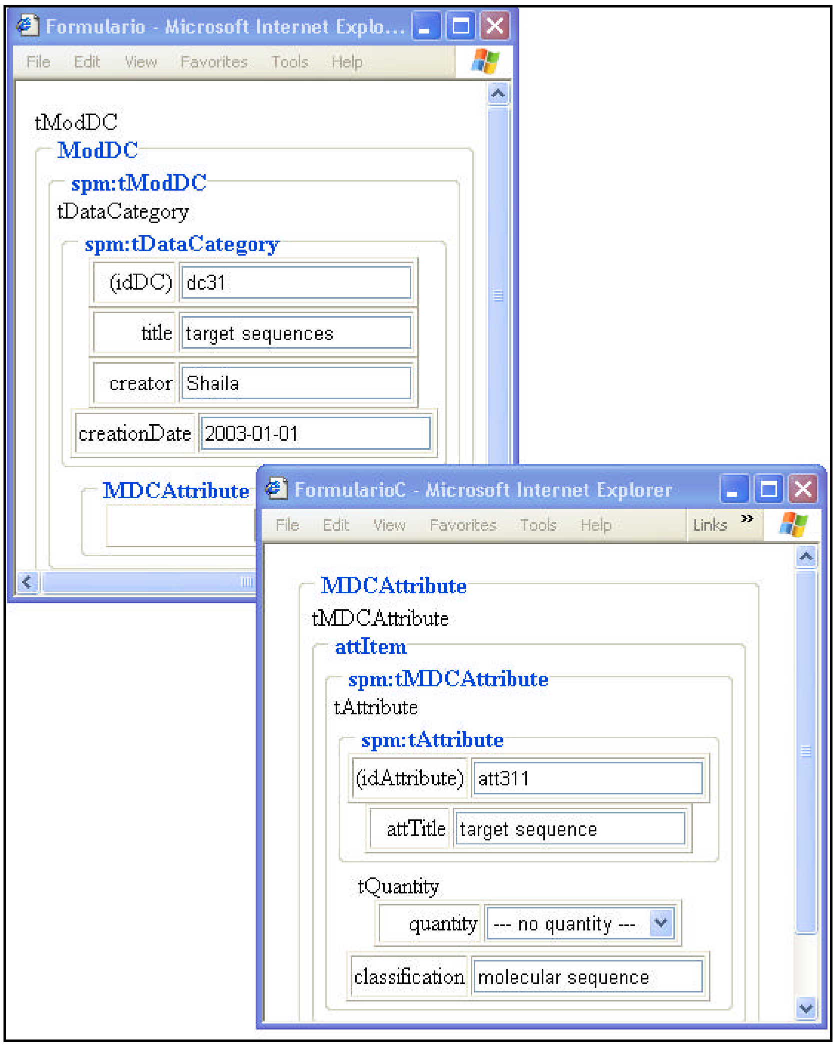
We begin by publishing the BLAST algorithm, as an algorithmic model. The BLAST algorithm (ALTSCHUL, S.F., GISH, W., MILLER, W., MYERS, E.W., LIPMAN, D.J., 1990) is a dynamic programming algorithm for pairwise sequence alignment. It is simple and robust and it can be implemented in a number of ways and applied to a variety of contexts including DNA and protein sequence database searches.

The idea behind algorithms of sequence alignment is to count on the empirical knowledge in molecular biology, i.e., when two molecules share similar sequences, they are also likely to share similar 3D structures and biological functions. Therefore, the similarity search usually counts on a database of sequences, against which it compares a target sequence (KANEHISA, M., 2000).

There is only one input for the BLAST model: a target sequence. Then, this input may be described as an *input model data category* composed of just one *model attribute*: sequence itself. In this case, there is no quantity involved but an object that must be represented when implemented. However, at the model level, representation details are not yet required. Instead, the essential is to establish the nature of the objects that are to be transformed or manipulated by the model. Considering BLAST algorithm was created to deal with molecular sequences, then, we have named a model data category "target sequences" which is composed of just one attribute called *target sequence*. As this attribute is not a quantifiable attribute, there is no quantity associated to it, and its *classification* is described as a "molecular sequence". Figure 80 shows how the scientist uses the SRMW Publication module interface to describe the Model Data Category for the *target sequences* input. This module is responsible for translating the information into the XML document shown in Figure 81, which is valid according to the SPMW XML Schema.

BLAST results correspond to a set of reference sequences, their alignments and scores. This output may be described as an *output model data category* for the BLAST model, with *model attributes* describing the reference sequence itself, and its corresponding alignment and score. These attributes are not associated to a quantity but all three are described according to their classification: the reference sequence is classified as a "molecule sequence"; the alignment of this reference sequence with the target sequence is classified as a “weight sequence”; and the score of this alignment is classified as a “similarity index”.

The BLAST algorithm is based on a systematic search of conserved words (KANEHISA, M., 2000). A word W is a sequence of letters of a limited size (e.g., 3 for amino acids and 11 for nucleotides). The user should determine the word size to guide the BLAST algorithm on the decomposition of the query sequence in words of that size. The resulting list of words is added with similar words, which are collected from all other combinations of word according to a given threshold T of similarity with the original word. Both W and T may be considered as parameters for the BLAST model, and therefore, we can publish them as part of a configuration model data category.


Formulario - Microsoft Internet Explo...
File Edit View Favorites Tools Help
FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help Links

tModDC
ModDC
spm:tModDC
tDataCategory
spm:tDataCategory
(idDC) dc31
title target sequences
creator Shaila
creationDate 2003-01-01
MDCAttribute

MDCAttribute
tMDCAttribute
attItem
spm:tMDCAttribute
tAttribute
spm:tAttribute
(idAttribute) att311
attTitle target sequence
tQuantity
quantity --- no quantity ---
classification molecular sequence

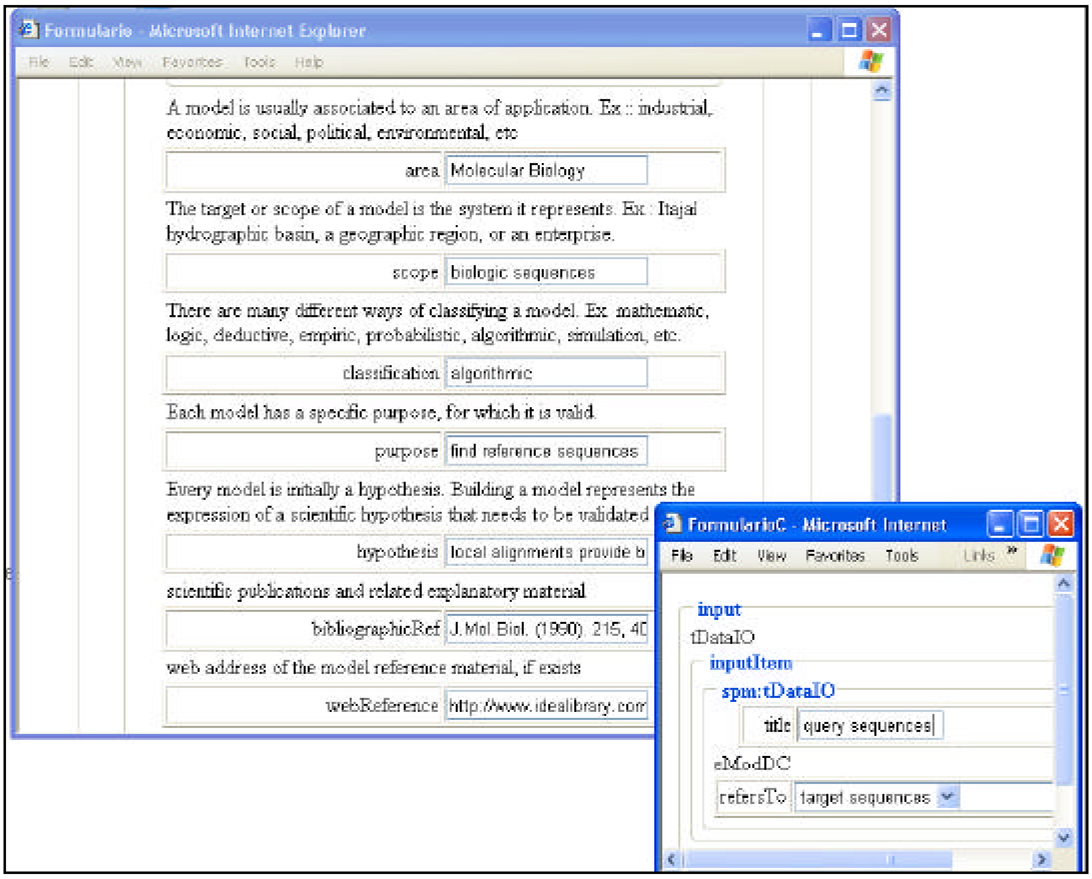
Figure 80: BLAST Algorithm Input Model Data Category Publication

```
<ScientificResourceDefinitions>
    <ModDC idDC=”dc31”>
        <title>Target Sequences</title>
        <creator>Shaila</creator>
        <creationDate>2003-01-01</creationDate>
        <MDCAttribute>
```

```
            <attItem idAttribute=”att311”>
                <attTitle>Target Sequence</attTitle>
                <quantity>no quantity</quantity>
                <classification>molecular sequence</classification>
            </attItem>
        </MDCAttribute>
    </ModDC>
    <ModDC idDC=”dc32”>
        <title>Configuration</title>
        <creator>Shaila</creator>
        <creationDate>2003-01-01</creationDate>
        <MDCAttribute>
            <attItem idAttribute=”att321”>
                <attTitle>Word</attTitle>
                <quantity>length</quantity>
                <classification>molecular sequence window </classification>
            </attItem>
        </MDCAttribute>
        <MDCAttribute>
            <attItem idAttribute=”att322”>
                <attTitle>Threshold</attTitle>
                <quantity>no quantity</quantity>
                <classification>similarity index</classification>
            </attItem>
        </MDCAttribute>
    </ModDC>
    <ModDC idDC=”dc33”>
        <title>Reference Sequences</title>
        <creator>Shaila</creator>
        <creationDate>2003-01-01</creationDate>
        <MDCAttribute>
            <attItem idAttribute=”att331”>
                <attTitle>Reference Sequence</attTitle>
                <quantity>no quantity</quantity>
                <classification>molecular sequence</classification>
            </attItem>
        </MDCAttribute>
        <MDCAttribute>
            <attItem idAttribute=”att332”>
                <attTitle>Alignment</attTitle>
                <quantity>no quantity</quantity>
                <classification>weight sequence</classification>
            </attItem>
        </MDCAttribute>
        <MDCAttribute>
            <attItem idAttribute=”att333”>
                <attTitle>Score</attTitle>
                <quantity>no quantity</quantity>
                <classification>similarity index</classification>
            </attItem>
        </MDCAttribute>
    </ModDC>
</ScientificResourceDefinitions>
```

Figure 81: Model Data Categories XML document for BLAST Algorithm

After publishing all needed data categories, the publisher may describe the BLAST model, and relate it to two *I/O data* and one *parameter*, which refer to the *input*, *output and configuration model data categories*, respectively, already published. Model description involves information about its scope, area, purpose, etc. Figure 82 shows how the scientist uses the SRMW Publication module interface to describe the BLAST Model, and the corresponding XML document is shown in Figure 83.


Formulario - Microsoft Internet Explorer
File Edit View Favorites Tools Help
FormularioC - Microsoft Internet
File Edit View Favorites Tools Links

A model is usually associated to an area of application. Ex :: industrial, economic, social, political, environmental, etc

area: Molecular Biology

The target or scope of a model is the system it represents. Ex : Itajaí hydrographic basin, a geographic region, or an enterprise.

scope: biologic sequences

There are many different ways of classifying a model. Ex: mathematic, logic, deductive, empiric, probabilistic, algorithmic, simulation, etc.

classification: algorithmic

Each model has a specific purpose, for which it is valid.

purpose: find reference sequences

Every model is initially a hypothesis. Building a model represents the expression of a scientific hypothesis that needs to be validated

hypothesis: local alignments provide b

scientific publications and related explanatory material

bibliographicRef: J.Mol Biol. (1990) 215, 40

web address of the model reference material, if exists

webReference: http://www.idealibrary.com

input

tDataIO

inputItem

spm:tDataIO

title: query sequences

eModDC

refersTo: target sequences

Figure 82: BLAST Algorithm Publication

```
 <ScientificResourceDefinitions>
      <Model idTF="tf3">
           <title>BLAST Algorithm</title>
           <creator>Altschul, S. F. et al.</creator>
           <creationDate>1990-01-01</creationDate>
           <input>
                <inputItem>
                      <title>BLAST Input</title><refersTo >dc31</refersTo >
                </inputItem>
           </input>
           <parm>
                <parmItem>
                     <title>BLAST configuration</title><refersTo>dc32</refersTo >
                </inputItem>
           </parm>
           <output>
                <outputItem>
                     <title>BLAST Output</title><refersTo>dc33</refersTo >
                </outputItem>
           </output>
           <area>Molecular Biology</area>
           <scope>Biologic Sequences </scope>
           <classification>algorithmic</classification>
           <purpose>to find reference sequences similar to a target
sequence</purpose>
           <hypothesis>local alignments provide better similarity results
</hypothesis >
           <bibliographicRef>Altschul, S. F. et al. “Basic Local Alignment Search
Tool” J.Mol.Biol. (1990) 215, 403-410</bibliographicRef>
           <webReference>http://www.idealibrary.com</ webReference>
      </Model>
</ScientificResourceDefinitions>
```

Figure 83: Model XML document for BLAST Algorithm

6.2.2 Publishing BLASTP program

BLASTP is an implementation of the BLAST algorithm, specifically created to deal with protein molecular sequences. One of its implementations is available at NCBI (NCBI, 2002). Now, let us consider that the project team needs to publish the BLASTP *program* within the SRMW architecture. First of all, a Web service for that program should be built. Then, while publishing the BLASTP program in SRMW, the WSDL file location is informed and extended through SPMW elements. Figure 84 shows the WSDL file for the BLAST program. Based on some information in this file, the SRMW Publication module builds its publication interface to capture the extended program description.

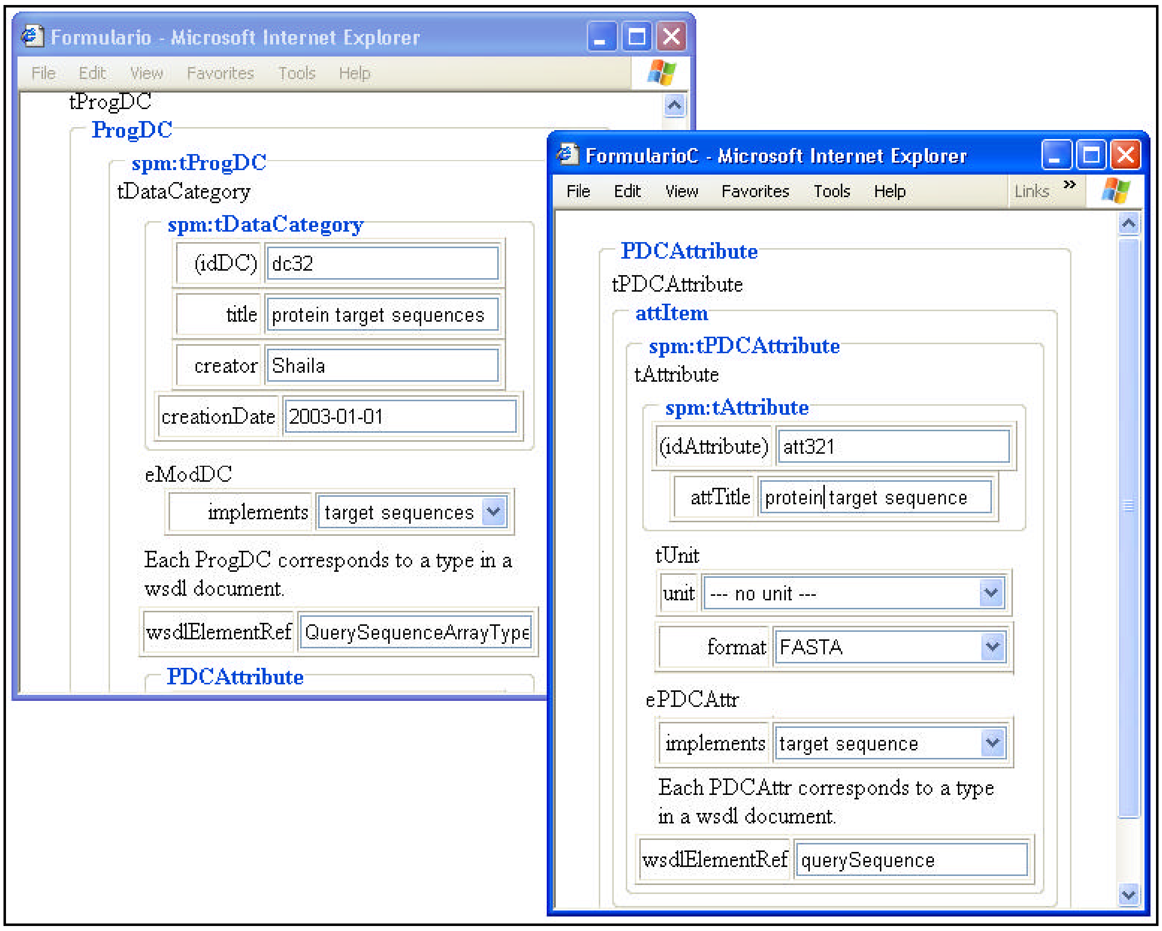
```
<definitions>
    <types>
        <xsd:schema>
            <xsd:complexType name="QuerySequenceArrayType">
                <xsd:complexContent>
                    <xsd:restriction base="soapenc:Array">
                        <xsd:attribute arrayType="xsd:QuerySequenceType[]"
                                    ref="soapenc:arrayType"/>
                    </xsd:restriction>
                </xsd:complexContent>
            </xsd:complexType>
            <xsd:complexType name="QuerySequenceType">
                <xsd:sequence>
                    <xsd:element name="querySequenceId" type="xsd:string"/>
                    <xsd:element name="querySequence" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="ConfigurationType">
                <xsd:sequence>
                    <xsd:element name="wordsize" type="xsd:integer"/>
                    <xsd:element name="threshold" type="xsd:float"/>
                    <xsd:element name="db" type="xsd:string"/>
                    <xsd:element name="expect" type="xsd:float"/>
                    <xsd:element name="matrix" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="HitArrayType">
                <xsd:complexContent>
                    <xsd:restriction base="soapenc:Array">
                        <xsd:attribute arrayType="xsd:HitType[]"
                                    ref="soapenc:arrayType"/>
                    </xsd:restriction>
                </xsd:complexContent>
            </xsd:complexType>
            <xsd:complexType name="HitType">
                <xsd:sequence>
                    <xsd:element name="hitSequenceId" type="xsd:string"/>
                    <xsd:element name="querySequence" type="xsd:string"/>
                    <xsd:element name="hitSequence" type="xsd:string"/>
                    <xsd:element name="score" type="xsd:integer"/>
                    <xsd:element name="e-value" type="xsd:float"/>
                    <xsd:element name="identities" type="xsd:float"/>
                    <xsd:element name="positives" type="xsd:float"/>
                    <xsd:element name="gaps" type="xsd:float"/>
                    <xsd:element name="alignSequence" type="xsd:string"/>
                    <xsd:element name="alignSize" type="xsd:integer"/>
                    <xsd:element name="querySeqOffset" type="xsd:integer"/>
                    <xsd:element name="hitSeqOffset" type="xsd:integer"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:schema >
        </types>
        <message name="codeInputmsg">
            <part name="configurationPart" type="ConfigurationType"/>
            <part name="dataPart" type="QuerySequenceArrayType"/>
        </message>
        <message name="codeOutputmsg">
            <part name="datatPart" type="HitArrayType"/>
        </message>
        <portType name="blastpPortType">
            <operation name="blastpOperation">
                <input message="tns:codeInputmsg"/>
                <output message="tns:codeOutputmsg"/>
            </operation>
        </portType>
    …
    </definitions>
```

Figure 84: WSDL document for BLASTP Program (abstract part)

Similarly to the model publication, the scientist should first describe the related BLASTP *program input/output data categories* that correspond to the BLAST model data categories, as shown in Figure 85. The BLAST model input data category *target sequences* corresponds to the BLASTP program input data category *protein target sequences*, which is composed of two *program attributes*. The *protein target sequence program attribute*, which corresponds to the *target sequence model attribute*, describes the target sequence as a string type written in the "FASTA alphabet" format. The other program attribute is added to carry the identity of the target sequence (*protein target sequence id*). It is described as a string type written in the PDB database identification format. The WSDL document, shown in Figure 84, already describes these attributes in terms of their primitive types, while the SPMW document, in Figure 87, is in charge of describing attribute formats and units, and also its association to the correspondent model attribute and WSDL description element. For instance, the program attribute *protein target sequence* is associated to the model attribute *target sequence*, and the WSDL element *querySequence*.

The BLASTP program output data category *protein reference sequences*, corresponds to the BLAST model output data category *reference sequences*, which is composed by three *program attributes*. The *hit sequence program attribute* corresponds to the *reference sequence model attribute*, and describes the reference sequence as a string type written in the "FASTA alphabet" format. The *align sequence program attribute* corresponds to the *weight sequence model attribute*, and describes the weight sequence as a string type written in a "similarity alphabet" format. The *score program attribute* corresponds to the *score model attribute*, and describes the score as an integer type expressed in "bits" unit. The BLASTP program output data category adds some other extra attributes, which give some useful information. For instance, the expect value (*e-value*) describes the number of hits one can "expect" to see just by chance when searching a database of a particular size. Other attributes, such as *identities*, *positives* and *gaps*, inform the percentage of character identity, similarity and gaps, respectively, between the query and hit sequences.

Also, the publisher should describe BLASTP parameters. A *configuration program data category* is also published, to correspond to the *configuration model data category*. Besides the attributes that correspond to the attributes at the model level, BLASTP configuration data category includes some extra attributes, such as the reference database. Therefore, with this implementation of the BLAST algorithm it becomes possible to choose the database against which to compare target sequences.


Formulario - Microsoft Internet Explorer
File Edit View Favorites Tools Help
FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help Links

tProgDC
ProgDC
spm:tProgDC
tDataCategory
spm:tDataCategory

(idDC)	dc32
title	protein target sequences
creator	Shaila
creationDate	2003-01-01

eModDC

implements	target sequences

Each ProgDC corresponds to a type in a wsdl document.

wsdlElementRef	QuerySequenceArrayType

PDCAttribute

PDCAttribute
tPDCAttribute
attItem
spm:tPDCAttribute
tAttribute
spm:tAttribute

(idAttribute)	att321
attTitle	protein target sequence

tUnit

unit	--- no unit ---
format	FASTA

ePDCAttr

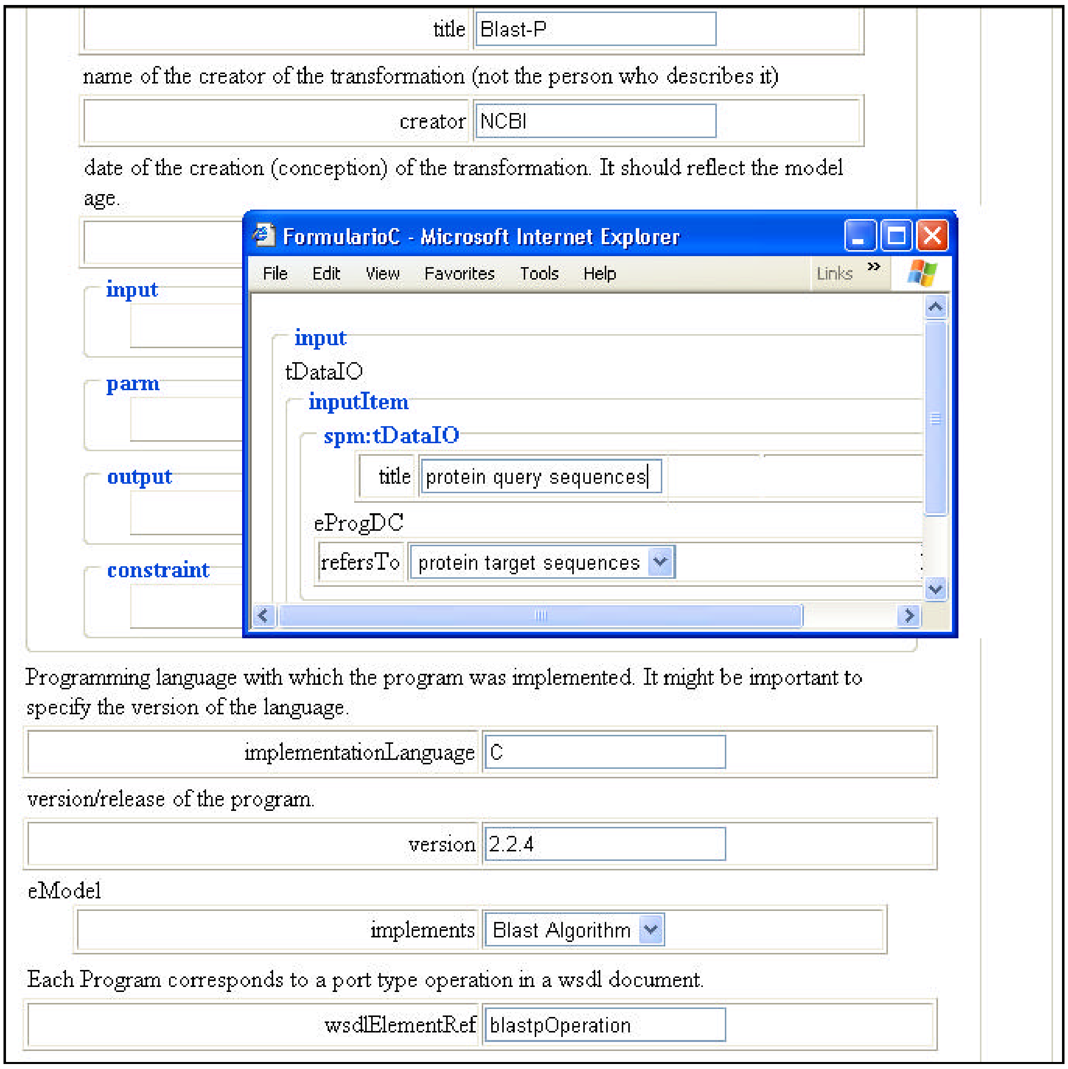
implements	target sequence

Each PDCAttr corresponds to a type in a wsdl document.

wsdlElementRef	querySequence

Figure 85: BLASTP Program Input Data Category Publication

After publishing program data categories, the publisher may describe the BLASTP *program*, and relate it to its *I/O data* and *parameters*, which relate to the *program data categories* previously published, as shown in Figure 86. Finally, the publisher may relate BLASTP *program* to the *model* it implements, the BLAST *model*.



title Blast-P

name of the creator of the transformation (not the person who describes it)

creator NCBI

date of the creation (conception) of the transformation. It should reflect the model age.

input
parm
output
constraint
FormularioC - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links

input

tDataIO

inputItem

spm:tDataIO

title protein query sequences

eProgDC

refersTo protein target sequences

Programming language with which the program was implemented. It might be important to specify the version of the language.

implementationLanguage C

version/release of the program.

version 2.2.4

eModel

implements Blast Algorithm

Each Program corresponds to a port type operation in a wsdl document.

wsdlElementRef blastpOperation

Figure 86: BLASTP Program Publication

```
<ScientificResourceDefinitions>
    <ProgDC idDC="dc32">
        <title>protein target sequences</title>
        <creator>Shaila</creator>
        <creationDate>2003-01-01</creationDate>
        <wsdlElementRef>QuerySequenceArrayType</wsdlElementRef>
        <PDCAttribute>
            <attItem idAttribute="att321">
                <attTitle>protein target sequence id</attTitle>
                <unit>no unit</unit>
                <format>PDB identification</format>
                <wsdlElementRef>querySequenceId </wsdlElementRef>
            </attItem>
        </PDCAttribute>
        <PDCAttribute>
            <attItem idAttribute="att322">
                <attTitle>protein target sequence</attTitle>
                <unit>no unit</unit>
                <format>FASTA alphabet</format>
                <wsdlElementRef>querySequence</wsdlElementRef>
            </attItem>
        </PDCAttribute>
    </ProgDC>
</ScientificResourceDefinitions>
<ScientificResourceDefinitions>
    <Program idTF="tf4">
        <title>Blast-P</title>
        <creator>NCBI</creator>
        <creationDate>2002-01-01</creationDate>
        <input>
            <inputItem>
                <title>protein query sequencestitle>
                <refersTo>dc32</refersTo>
            </inputItem>
        </input>
        <parm>
            <parmItem>
                <title>Blast-P Configuration</title>
                <refersTo>dc33</refersTo>
            </parmItem>
        </parm>
        <output>
            <outputItem>
                <title>hit sequences</title>
                <refersTo>dc34</refersTo>
            </outputItem>
        </output>
        <implementationLanguage>C</implementationLanguage>
        <version>2.2.4</version>
        <implements>tf3</ implements>
        <wsdlElementRef>BlastpOperation</wsdlElementRef>
    </Program>
</ScientificResourceDefinitions>
```

Figure 87: Program and ProgDC XML document for BLASTP program

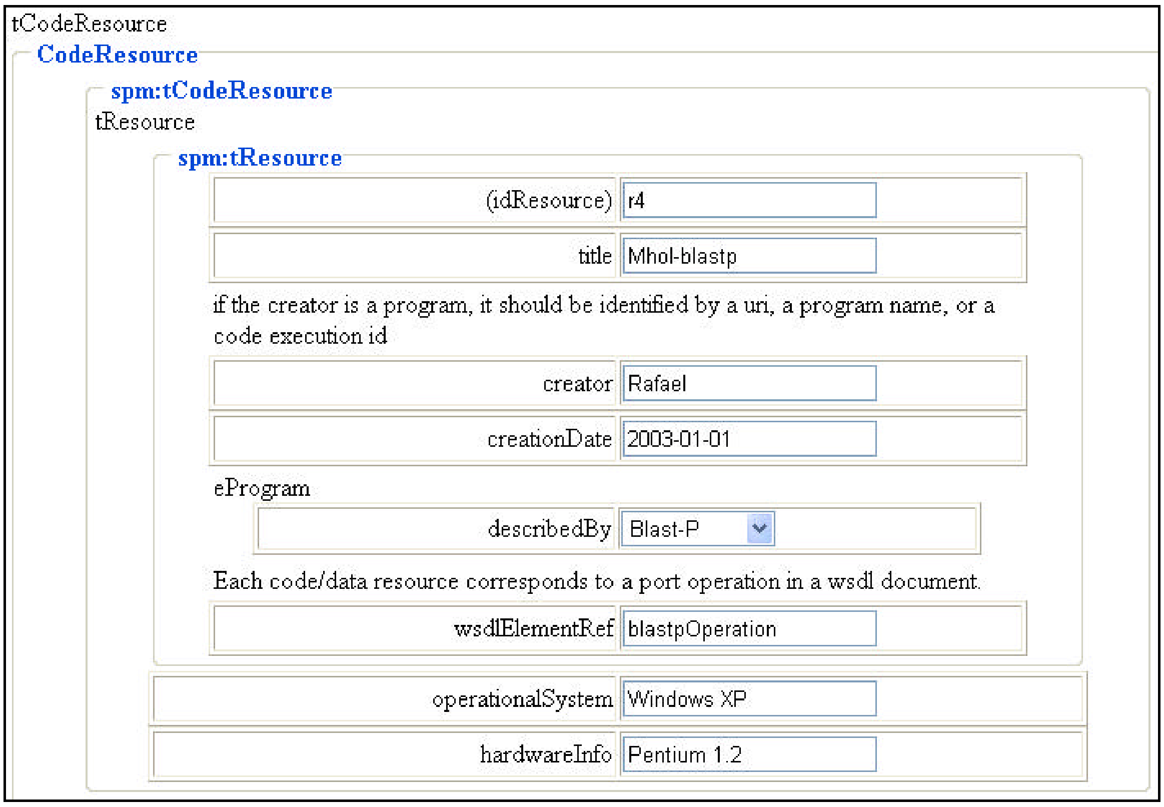
6.2.3 Publishing MHOL-BLASTP code resource

After publishing the BLASTP program interface it is now possible to publish a related code resource. This compiled code is available for execution as a Web service, and its concrete description is presented in Figure 88. Similarly to the Program publication, SPMW provides description elements for extending the WSDL contents. The SRMW Publication module interface, presented in Figure 89, shows how to associate the code resource to its correspondent WSDL Port Type Operation and SPMW Program elements.

```
<definitions>
…
    <binding name="blastpBinding" type="tns:blastpPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="blastpOperation">
            <soap:operation soapAction="capeconnect::blastpPortType#NewOperation"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://www.your-company.com/blastp/binding"
use="encoded"/>
            </input>
            <output>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://www.your-company.com/blastp/binding"
use="encoded"/>
            </output>
        </operation>
    </binding>
```

```
        <service name="blastp">
            <port binding="tns:blastpBinding" name="blastpPort">
                <soap:address location="http://localhost:8000/ccx/blastp"/>
            </port>
        </service>
    </definitions>
```

Figure 88: WSDL document for BLASTP Program (concrete part)



tCodeResource
CodeResource
spm:tCodeResource
tResource
spm:tResource

(idResource)	r4
title	Mhol-blastp

if the creator is a program, it should be identified by a uri, a program name, or a code execution id

creator	Rafael
creationDate	2003-01-01

eProgram

describedBy	Blast-P

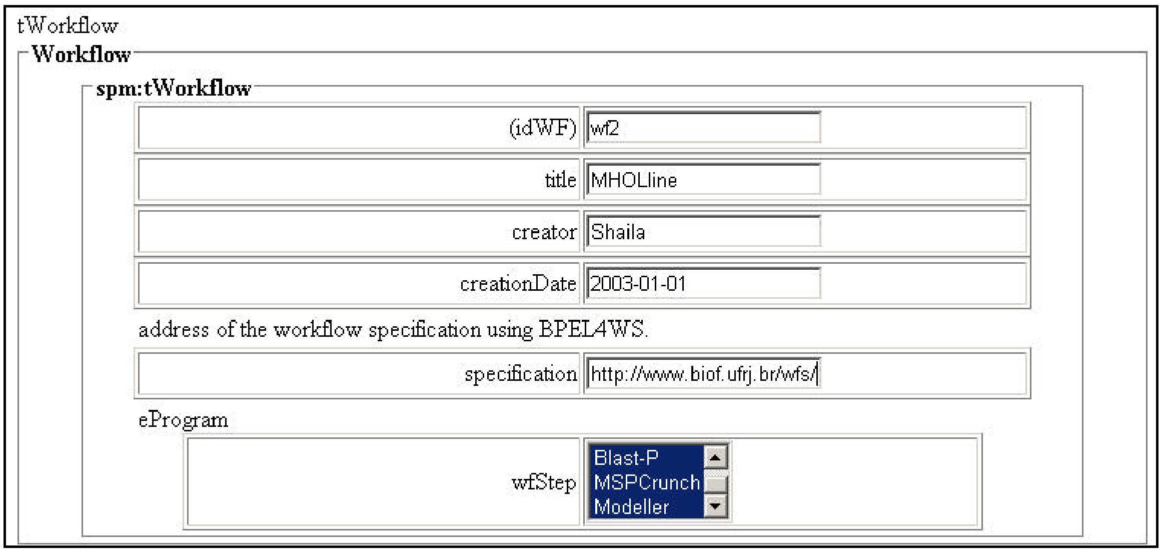
Each code/data resource corresponds to a port operation in a wsdl document.

wsdlElementRef	blastpOperation
operationalSystem	Windows XP
hardwareInfo	Pentium 1.2

Figure 89: MHOL-blastp Code Resource Publication

6.2.4 Publishing MHOLline Workflow

As we already explained, BLASTP and other programs are part of a specific Workflow called MHOLLine. To make the MHOLline workflow available for experiments, we need to publish it. A Workflow edition tool helps on the specification of the MHOLline workflow, which is then informed to the Publication module, as shown in Figure 90. Then the Publication model stores the workflow description as the XML document presented in Figure 91.


tWorkflow
Workflow
spm:tWorkflow

(idWF)	wf2
title	MHOLline
creator	Shaila
creationDate	2003-01-01

address of the workflow specification using BPEL4WS.

specification	http://www.biof.ufrj.br/wfs/

eProgram

wfStep	Blast-P MSPCrunch Modeller

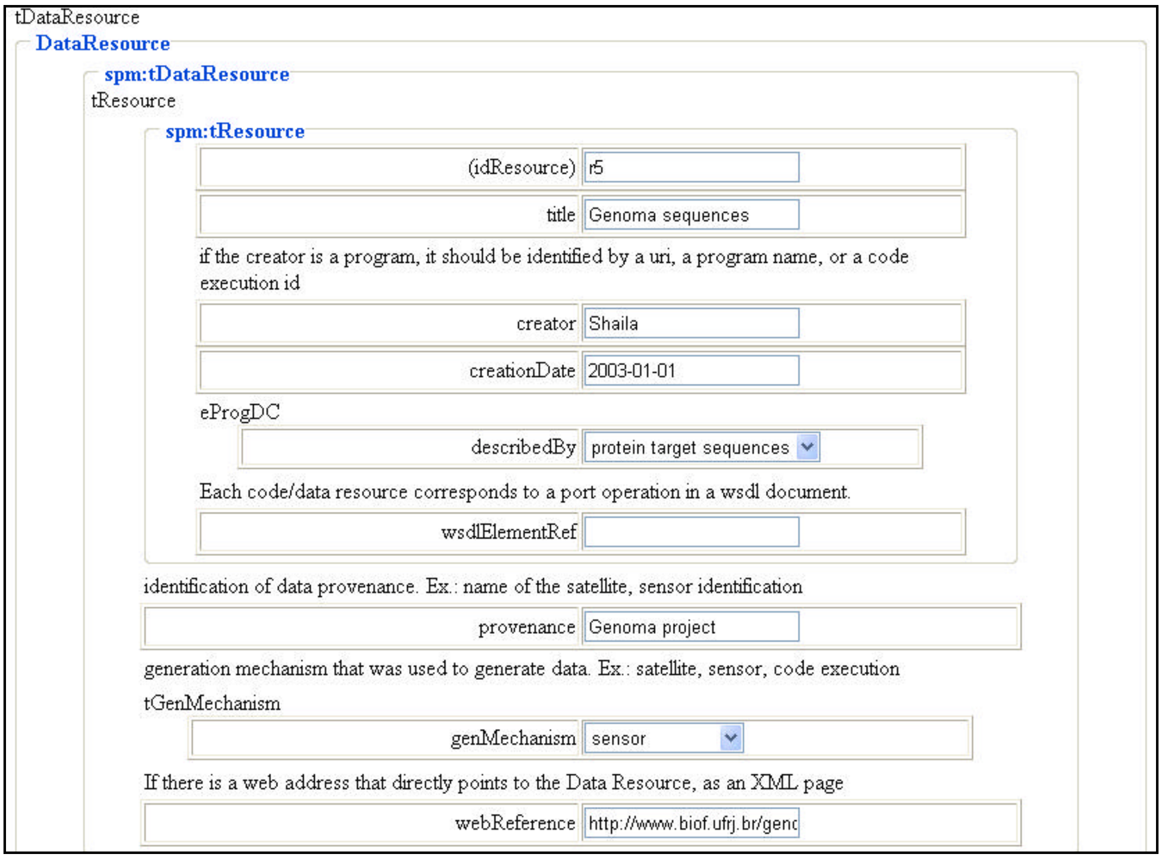
Figure 90: MHOLline Workflow Publication

```
<ScientificResourceDefinitions>
<Workflow idWF="wf2">
    <title>MHOLline </title>
    <creator>Shaila</creator>
    <creationDate>2003-01-01</creationDate>
    <specification>http://www.myWfDefs.br/mholline.bpel</specification>
</Workflow >
</ScientificResourceDefinitions>
```

Figure 91: MHOLline Workflow XML document

6.2.5 Publishing Genoma data resources

In fact, MHOLline workflow is still under evaluation. Therefore, to prove its efficiency, a set of Genoma sequences is used to test it. To have these data available as a scientific resource in SRMW, the scientist needs to publish them as a Web service. Also, while publishing these data, to use them as input to the MHOLline workflow requires that their type definition be compatible to the type definition referred by the BLASTP program input, i.e., the protein target sequences data category. Figure 92 shows how the Cabiunas data resource is described, and how it is associated to its correspondent program data category and WSDL Port Type Operation elements.


tDataResource

DataResource

spm:tDataResource

tResource

spm:tResource

(idResource)	r5
title	Genoma sequences

if the creator is a program, it should be identified by a uri, a program name, or a code execution id

creator	Shaila
creationDate	2003-01-01

eProgDC

describedBy	protein target sequences

Each code/data resource corresponds to a port operation in a wsdl document.

wsdlElementRef	

identification of data provenance. Ex.: name of the satellite, sensor identification

provenance	Genoma project

generation mechanism that was used to generate data. Ex.: satellite, sensor, code execution

tGenMechanism

genMechanism	sensor

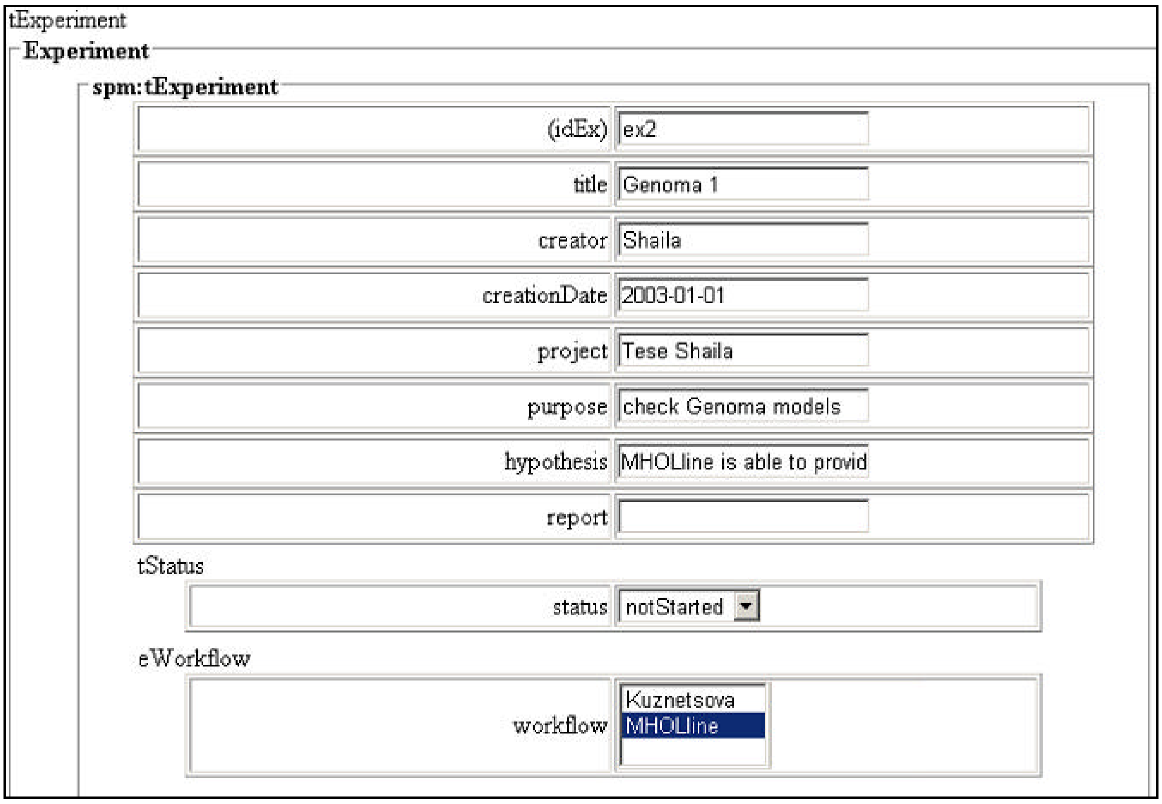
If there is a web address that directly points to the Data Resource, as an XML page

webReference	http://www.biof.ufrj.br/genc

Figure 92: Genoma Data Resource Publication

6.2.6 Publishing the Genoma Experiment

To start a new experiment in SRMW the scientist first publishes it. Figure 93 shows how to describe the Genoma experiment, and how to associate it to the workflows it is supposed to be using during essays. The Experiment XML document is presented in Figure 94. Note that the experiment document does not show any essays, meaning it has not started yet.


tExperiment

Experiment

spm:tExperiment

(idEx)	ex2
title	Genoma 1
creator	Shaila
creationDate	2003-01-01
project	Tese Shaila
purpose	check Genoma models
hypothesis	MHOLline is able to provid
report	

tStatus

status	notStarted

eWorkflow

workflow	Kuznetsova MHOLline

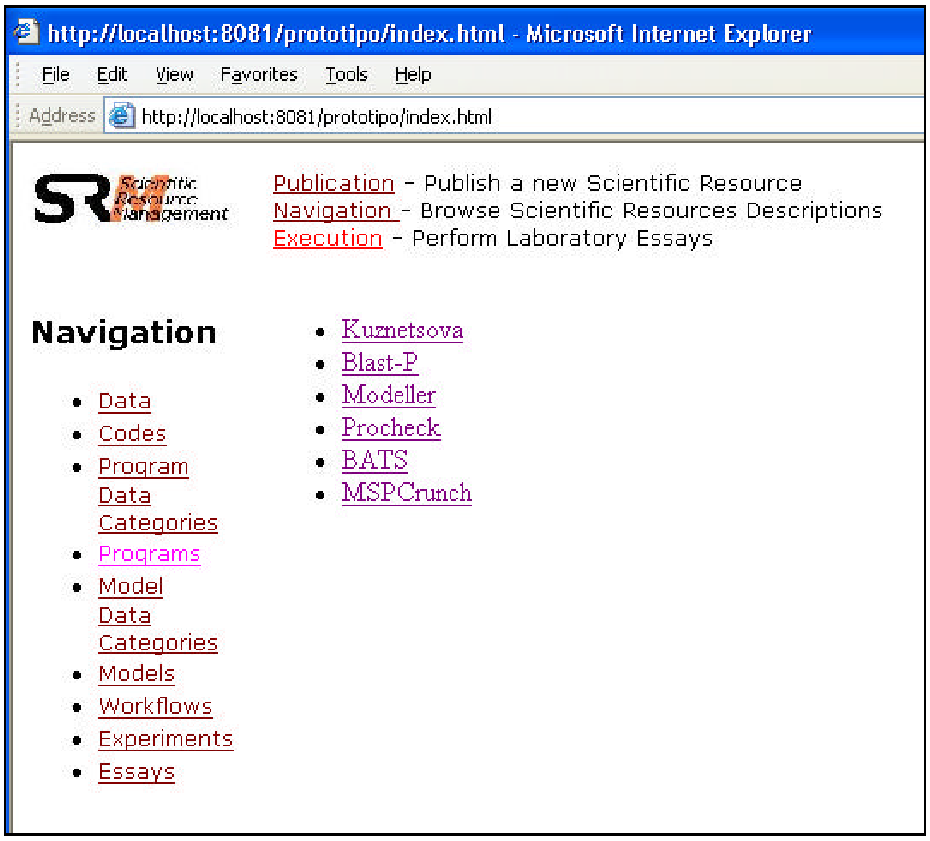
Figure 93: Publishing Experiments using the MHOLline workflow

```
<ScientificResourceDefinitions>
<Experiment idEx ="ex2”>
     <title>Genoma 1</title>
     <creator>Shaila</creator>
     <creationDate>2003-01-01</creationDate>
     <project>Tese Shaila</project>
     <purpose>check Genoma models</purpose>
     <hypothesis>MHOLline is able to provide good quality 3D models</hypothesis>
     <report></report>
     <status>notStarted</status>
     <workflow >wf2</workflow >
</Experiment>
</ScientificResourceDefinitions>
```

Figure 94: Genoma Experiment XML document

6.2.7 Navigating through Structural Genomic Resources

Frequently, a structural genomic project team receives requests to generate 3D models for a set of molecule sequences. Let us consider a structural genomic workflow could have been defined in terms of scientific models (a model-based workflow), having MHOLline workflow as a program-based workflow that implements it. Considering MHOLline workflow may not be adequate to a specific request, a new workflow should be built. Suppose, for instance, BATS program, which is a step of MHOLline workflow, is not suitable for this specific request. Having access to the model-based workflow description, the scientist would be able to browse the programs that implement each step (model) of this workflow. Selecting the BLAST filter step at the model-based workflow description, the scientist realizes an alternative to BATS would be to use MSPCrunch program. Then, through browsing the available programs (Figure 95), and their corresponding data categories, the scientist can build an adequate program-based workflow for the new experiment.


http://localhost:8081/prototipo/index.html - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8081/prototipo/index.html

Publication - Publish a new Scientific Resource
Navigation - Browse Scientific Resources Descriptions
Execution - Perform Laboratory Essays

Navigation

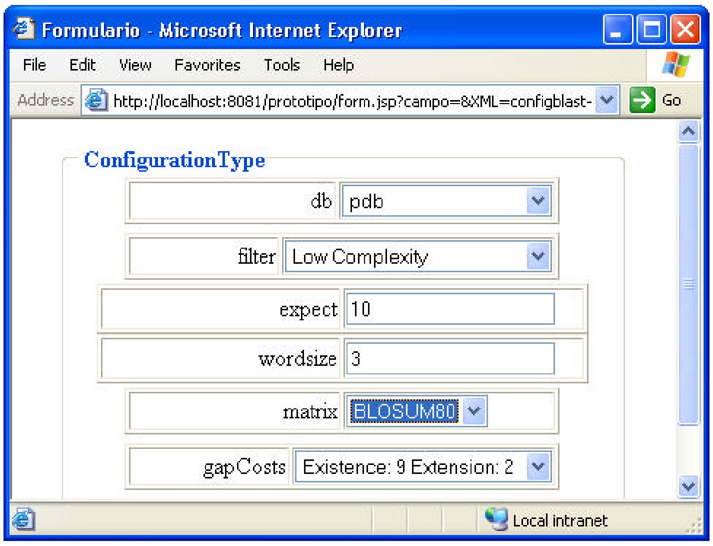
- Data
- Codes
- Program Data Categories
- Programs
- Model Data Categories
- Models
- Workflows
- Experiments
- Essays

- Kuznetsova
- Blast-P
- Modeller
- Procheck
- BATS
- MSPCrunch

Figure 95: Navigating through Programs

6.2.8 Executing the Genoma Experiment

After publishing an experiment, the specialist may then start it, by providing all program input data and parameter values. Considering BLASTP has parameter values to be set, and which were not previously published as a data resource, SRMW helps the user on setting them up on the fly, as shown in Figure 96.


Formulario - Microsoft Internet Explorer

http://localhost:8081/prototipo/form.jsp?campo=&XML=configblast-

ConfigurationType

db	pdb
filter	Low Complexity
expect	10
wordsize	3
matrix	BLOSUM80
gapCosts	Existence: 9 Extension: 2

Figure 96: BLASTP program parameter input Interface

In addition, the Experimentation module helps the user on choosing the adequate input data, considering only those data resources that are described by the same BLASTP program input data category. In this case, the Genoma data will be an option. Finally, the scientist is now able to “execute” the experiment.

Once the experiment results confirm the experiment hypothesis, i.e., that MHOLline provides good quality 3D models; the specialist decides to finish the Genoma experiment. During the experiment, the Experimentation module automatically updates the Genoma experiment document, registering all the related essays. Then, the user may choose to finish the experiment, and completes the diagnosis report for that experiment. The Genoma experiment will now be available for other scientists, who will find it through the Navigation module.

6.3 Final Considerations

In SRMW, users have to include a large amount of scientific information before starting to benefit from them. Despite this, after a while, the scientific activity is facilitated, and users have just to fill-up parameters and provide data resources. As a consequence, scientific resources get documented, integrated, inter-related and consistent.

Those two applications have helped us to confirm some modelling concepts such as models and programs, and more importantly, helped us to extend the semantic levels of resources including essays and experiments.

The main benefit of this approach is on the organization of scientific resources to enable their management. Those two applications have evidenced SRMW facilities such as:

?? Code resources can be browsed according to the associated model or program.

?? Users can see clearly parameter and input distinction, becoming conscious of how to “tune” the program in use.

?? Model and program distinction reflects the different levels of abstraction, allowing for a more selective browsing.

?? Identification of constraints at the model level, allowing users to discard inadequate models when searching for one.

?? The explicit representation of constraints at the program level enables the verification of data or parameter input before actually accessing and executing the code.

?? Identification of scientific experiments, and their organisation as a set of essays.

?? Browsing documented model usage.

?? Explicit representation of program IO so it becomes easier to combine programs in a scientific workflow.

7. Conclusion

In this thesis we have focused on scientific resource management. We have deeply analysed several scientific applications trying to understand their current problems with respect to database technology issues.

We have first tried to organise scientific data through high level metadata, which has been the focus of database community for the last decades. However, our first understanding of the problem revealed that in the scientific area, data is not the only resource to be represented and managed through DBMS technologies. In fact, it does not work if handled separately, in the same way as business systems are being developed today. As a matter of fact, the 1998 Asilomar report (BERNSTEIN, P. *et al.*, 1998) has defined the management of programs and data as one of the challenges that the database research community had to face for the ten years to follow that report. As a result from the detailed analysis of applications described in Chapter 2, we defined not only programs and data, but also, models, workflows and experiments.

To help on the definition of the main problems related to scientific resources management we specified three main requirements common to most scientific applications, i.e., (i) how to handle their heterogeneity and distribution, (ii) how to describe them, (iii) how to combine them and register their use. To facilitate the resource exchange among scientists, a support environment should organize, manage and monitor these resources, providing interoperability, reusability and flexibility.

To address those requirements we designed SRM architecture, defined SPM metamodel and implemented a prototype taking advantage of the recent Web services technology. SRM aims to address the management problems currently faced by the scientific community. The main goal of SRM is to provide metadata support for managing distributed scientific resources. Its main component is the Scientific Publishing Metamodel (SPM) specially designed to describe scientific resources. Considering Web services as a platform independent infra-structure, which provides tools that facilitate the operation of an *in silico* laboratory, we have implemented a prototype of the SRM architecture based on Web services (SRMW). SPM we have implemented as a XML Schema (SPMW) to be instantiated as an extension of WSDL documents. Finally, we have shown the adequacy of SPMW in the context of two different scientific research teams.

7.1 Contributions

The development of SRM, SPM, SRMW prototype and its experimentation has led to several contributions. The first and main contribution of this thesis is the SPM metamodel. SPM innovates by providing several different semantic levels of scientific resources representation, while integrated to a scientific resources management system. Similarly to our work, metamodel based architectures have also been proposed to address the scientific community (FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003), (ESP2NET PROJECT), (ESSW PROJECT), (BENZ, J.; HOCH, R.; GABELE, T., 1997). However, their metamodels lack important scientific description support, particularly with respect to model and program distinction. Other interesting metadata support approaches try to establish domain standards through the proposal of domain ontologies, specifically in the genetic area (WROE, C. *et al.*, 2003) (CRITCHLOW, T.; MUSICK, R.; SLEZAK, T., 2001). However, we believe that a generic scientific approach such as ours is more suitable when one cannot count on available domain ontologies for each scientific area. In addition, we believe that explicit representation of metadata as occurs in metamodel approach improves comprehension. A recent work (SPYNS, P, MEERSMAN, R., JARRAR, M., 2002) identifies advantages in both metamodel and ontology approaches and proposes to combine them.

The second contribution of this work is the set of SPM concepts specially designed to represent scientific resources, enabling their management. First of all SPM includes an explicit semantic representation of scientific models, providing their representation at both theoretic and operational levels. Codes, programs and models are represented by different concepts, each one at a different level of abstraction. Analogously, the same occurs to data category concepts. Data input and parameters are also separated concepts, as each one plays a different role at the program or model they are associated to. Finally, SPM includes concepts to represent scientific workflows, essays and experiments, allowing for the registry of the use of scientific resources. All these concepts were analysed in the light of two real applications, with the collaboration of scientists from the Petrobras Research Centre (CENPES), and from the UFRJ Biophysics Institute (IBCCF).

A third contribution of this work is the SRM architecture, which allows scientific users to browse, publish and execute scientific programs and workflows. In addition, SRM also provides the automatic registry of these executions, as part of essays and experiments, which can be analysed along those several levels of semantic abstractions. This is an important step towards data provenance and derivation.

Another contribution can be derived through the use of Web services for scientific applications. We believe that the use of Web services is a first and fundamental step in the direction of a full featured *in silico* laboratory as they provide the required interoperability to support *in silico* experiments. They allow scientists to dynamically publish, discover, and aggregate scientific resources through the Internet. Scientific and business communities are just beginning to use this new technology. If compared to other approaches, the Web services approach is superior with regard to interoperability, reusability and flexibility issues. It overcomes platform incompatibilities among software tools and databases, and orchestrates their interaction. For instance, MHOLline workflow presented in section 6.2 includes Web services that interface with legacy programs written in different languages (e.g., Fortran and C). In addition, the Web services workflow definition language provides more flexibility than scripts as it allows e-scientists to build *ad hoc* service compositions. Also, since these compositions run through Web services, additional registering and documentation can be included on the fly, helping data provenance and automatic execution documentation. Reusability is facilitated by Web services because of their modular characteristic. Web services workflows are also published as Web services, and this enables other scientists to use them as part of new service compositions. Furthermore, Web services are an open standard already adopted by the industry, and therefore their approach is not tied to any proprietary solution.

A fifth contribution of this work is the publication of scientific metadata through Web services, using the SPMW schema. Web services architecture is not enough to provide a full-featured *in silico* laboratory. We have reviewed similar approaches that address distribution and heterogeneity of data and programs (FOSTER, I., KESSELMAN, C., *et al.*, 2002) (LESELECT), but these initiatives are also not sufficient if not associated to an efficient metadata support. In the case of Web services, since its description language (WSDL) was originally proposed for generic service description, it lacks application-related semantic descriptors. SPMW uses the WSDL extensibility mechanisms to add higher level descriptors specifically related to scientific resources. For instance, SPMW documents provide dynamic molecular scientists with information about an *in silico* experiment, where a FASTA format molecular sequence was used as input to a BLASTP program based on the BLAST algorithm. WSDL extensibility enforces the adequacy of Web services to provide metamodel based solutions.

Finally, the SRMW prototype is another contribution of this work. Portability, interoperability and flexibility are some of the main benefits when managing distributed scientific resources through SRMW. As XML has become an international accepted standard to describe data resources, we have chosen to have our metadata expressed as XML documents based on the SPMW XML Schema. To attend specific scientific domain needs, SPMW was designed to be extensible, and extensibility elements are included in the schema. Moreover, SRMW has been designed to be flexible when new extensions to SPMW are included. To store SPMW documents, SRMW includes a metadata repository. We have developed interfaces to store these documents in the RDBMS MySQL, due to its wide use in Web applications. Finally, SRMW has been implemented using Java servlets, and includes IBM BPWS4J 1.0.1 (BPWS4J) to define and process workflow specifications, written in BPEL4WS (CURBERA, F., GOLAND, Y, *et al.*, 2002). Other applications may interact with it by issuing XML queries and handling XML documents resulting from such queries.

Other application areas, such as business-to-business (B2B) integration, can also benefit from this approach to enhance metadata semantics. For instance, the SPMW metamodel could be adapted to address B2B applications, including new descriptors. In this context, some descriptors could make use of existing vocabulary standards like the ones used in RosettaNet (ROSETTANET), to address higher information interoperability issues of terminology and data meaning.

7.2 Current and Future Work

The present work was motivated by two research projects in the environmental area: a CT-Petro project called SIMBIO (MOURA, F.A., 2001) and an INRIA-CNPq collaboration project called ECOBASE (ECOBASE, 2001). Both projects allowed us to interact with researchers seeking solutions to similar environmental problems.

In the context of the ECOBASE project, we have used Le Select (LESELECT) as the infrastructure for dealing with distribution and heterogeneity of scientific programs and data. However, once Le Select focus was not on metadata support, we have proposed SRM as a Le Select extension, providing mechanisms to capture model descriptions and to monitor the actual distributed usage of models, programs and data. Then, we have decided to redesign our SRM architecture by using Web services, so we could take advantage of W3C well-accepted standards, which would facilitate the use of workflow web engines based on these standards. Projects like ECOBASE bring a significant contribution to our research team, and we are already applying for a new similar project with focus on Web services composition.

In the context of SIMBIO project, we have studied models and experiments from the Cabiunas case study developed at CENPES, which has helped us to identify the concepts behind the scientific modelling context. Modellers and model users deal with models in a very subjective way. Indeed, in the Cabiunas case study, we could witness CENPES researchers successfully applied the Kuznetsova model out of its original scope. Having previous experiments documented and scientific model usage described, allowed researchers to apply a reservoir phenomena model to a surface phenomena, despite the completely different physic-chemical conditions. Future cooperation projects with CENPES are planned to develop SRMW experiments and SPMW refinements.

Recently, we have started a new informal project in the molecular biology area in collaboration with the UFRJ Biophysics Institute, particularly with Paulo Bisch, the head of Rio de Janeiro´s Bioinformatics Virtual Institute, and Shaila Rössle, a research assistant. We are currently analysing an application on the dynamic molecular biology domain, called MHOLline. This application has been particularly useful to help us on the identification of experiment and essay concepts. We have identified the need for workflow management and the need for keeping a record of intermediary results.

The work with Paulo Bisch´s team is being developed in association to other dissertations and thesis at COPPE Sistemas. The investigation of the difficulties on building a biological workflow using Web services is the focus of a master's dissertation work (TARGINO, R., 2003). Another dissertation work (TEIXEIRA, F., 2003) is concentrating on the development of facilities to support the generation of scientific data resources as Web services. Finally, a third dissertation work (GONÇALVES, F., 2003) has been developing an integration tool to support the publication of datasets associated to different data categories, allowing them to be integrated to different scientific programs, and take part on different workflows and experiments.

The SRM architecture can be applied not only to biocorrosion and molecular biology areas, but it also can be applied to many other scientific areas. Publishing models and experiments from scientists of different scientific areas has been helping us to identify the concepts behind the scientific modelling context. However, to build a scientific metamodel is not an easy task. Therefore SPM metamodel should be incrementally refined. Constraints, data replica, and workflow versions are some of the concepts that need to be included or developed in SPM.

Although we can not consider the examples presented in sections 6.1 and 6.2 as software evaluation studies, we could say that they are a first step on that direction. According to the classification used by the software engineering community (KITCHENHAM, B., PICKARD, L., PFLEEGER S., 1995) we have started a *blocked subject-project study*, by examining SRMW across a set of teams and a set of projects. However, we plan to focus on *replicated-project studies*, observing what is happening on different teams of one typical project. More specifically, our first case study is already in course at the Biophysics Institute.

As a result of our first publications on this work, we had made some initial contacts with international projects, while attending the Workshop on Data Derivation and Provenance (BUNEMAN, P., FOSTER, I., 2002), such as GriPhyN, at the University of Chicago, USA, and MyGrid, at the University of Edinburgh, Scotland, both working on metadata issues for the scientific community. It is our intention to further enlarge these contacts in order to continue our work.

MyGrid project (WROE, C. *et al.*, 2003) has been developing an interesting work on the use of domain-ontologies to support scientific applications in the genetic area. From our point of view, to build domain-ontologies for scientific applications is not an easy task. Most works (BOZSAK, E. *et .al,*, 2002) (NOY, N.F.; FERGERSON, R. W.; MUSEN, M. A., 2000) concentrate on building tools and representation frameworks for ontologies. However, there is a yet not very explored field on how to support the conceptualization process. A doctor’s thesis (GARCIA, S., 2003) at DCC/IM/NCE - UFRJ is currently developing studies on this direction. In addition, as a complement to SRM architecture, it is our intention to include the use of domain ontologies, to help scientists on searching for adequate resources, browsing their descriptions, while keeping our metamodel as the basis to register and query experiments.

Grid computing systems usually involve machine clusters and parallel processing. Metadata to support these systems should include information about the facilities they provide. GriPhyN project is already working in this direction, proposing a metamodel that provides support for the data replica management. SRM can also be extended to address Grid systems. In order to do this, we are investigating the use of Grid systems at IBCCF, as part of a doctor's thesis study (MEYER, L. A., 2003) at COPPE Sistemas.

As discussed in section 3.4, the support for workflow dynamic definition is a scientific workflow management needed facility. SRMW is not yet supporting this facility, but it is our intention to work on this issue also. In this direction, the SPM metamodel should provide support for workflow versioning control, such as the one provided in WASA system (WESKE, M.; VOSSEN, G.; MEDEIROS, C., 1996).

Despite the many advantages of the Web services technology, we believe it is important to consider it as a new and evolving technology. Therefore, another interesting research direction is to work on higher level abstractions for specifying scientific workflows. The goal of such abstractions is the decoupling of the workflow specification (and knowledge) from any specific technology. In this direction, it is our intention to work on issues such as quality evaluation techniques to support choosing equivalent workflow steps while executing it (CARDOSO, L. F., SOUZA, J. M., MARQUES, C., 2002) (AZEVEDO, V., PIRES, P., MATTOSO, M., 2003) and scientific model based workflow definitions.

8. References

AGUIRRE, L. A., 2000, **Introdução à Identificação de Sistemas: Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais**. UFMG, Belo Horizonte, Brazil (in portuguese).

AILAMAKI, A., IOANNIDIS, Y., LIVNY, M., 1998, "Scientific Workflow Management by Database Management". In: **Proceedings of 10th International Conference on Scientific and Statistical Database Management**, pp. 190-199, Capri, Italy, July.

ALLCOCK, B., FOSTER, I., NEFEDOVA, V., CHERVENAK, A., DEELMAN, E., KESSELMAN, C. LEE, J., SIM, A., SHOSHANI, A., DRACH, B. WILLIAMS, D., 2001, "High-Performance Remote Access To Climate Simulation Data: A Challenge Problem for Data Grid Technologies". In **Proceedings of Super Computing 2001**, Denver, USA, November.

ALTSCHUL, S.F., GISH, W., MILLER, W., MYERS, E.W., LIPMAN, D.J., 1990, "Basic local alignment search tool", **Journal of Molecular Biology**, v.215, pp. 403-410.

ALUR, R., HENZINGER, T. A., MANG, F., QADEER, S., RAJAMANI, S., TASIRAN, S., 1998, "MOCHA: Modularity in Model Checking". In **Proceedings of CAV 1998**, pp. 521-525.

ANKOLEKAR, A., BURSTEIN, M., HOBBS, J., LASSILA, O., MARTIN, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PAYNE, T., SYCARA, K., ZENG, H., 2001, "DAML-S: Semantic Markup for Web Services". In **Proceedings of the International Semantic Web Working Symposium** (SWWS), July-August.

ASHBURNER M., LEWIS S., 2002, "On Ontologies for Biologists: the Gene Ontology - uncoupling the web". In: **In Silico Biology**, Novartis Found Symposium, v.247, pp.66-83.

APACHE, AXIS Engine. Available from <http://xml.apache.org/axis/>

AZEVEDO, V., PIRES, P., MATTOSO, 2003, "Handling Dissimilarities of Autonomous and Equivalent Web Services". To appear in **Proceedings of the Workshop on E-business and Semantic Web**, Austria, June.

BANERJEE, S., BASU, A., 1993, "Model Type Selection in an integrated DSS environment", **Decision Support Systems**, vol. 9, pp.75-89.

BENZ, j., HOCH, R., LEGOVIC, T., 2001, "ECOBAS – modelling and documentation", **Ecological Modelling**, v.138, pp. 3 – 15.

BENZ, J., HOCH, R., GABELE, T., 1997, "Documentation of Mathematical Models in Ecology – An Unpopular Task?". In **ECOMOD Newsletter**, ISEM, December. Available from <http://ecomod.tamu.edu/ecomod/isem.html>

BENZ, J., HOCH, R., 1999, "**ECOBAS – Model Interchange Format Reference Manual**" – ECOBAS_MIF version 3.0.
Available from <http://dino.wiz.uni-kassel.de/ecobas/syntax_mif/syntax2_mif.ps> or from <ECOBAS_MIF version 3.1, http://eco.wiz.uni-kassel.de/ecobas/ecobas_mif.html>

BERNSTEIN, P., BRODIE, M., CERI, S., DEWITT, D., FRANKLIN, M., GARCIA-MOLINA, H., GRAY, J., HELD, J., HELLERSTEIN, J., JAVADISH, H., LESK, M., MAIER, D., NAUGHTON, J. PIRAHESH, H., STONEBRAKER, M., ULLMAN, J. , 1998, "The Asilomar Report on Database Research", **SIGMOD Record,** vol. 27, no. 4.

BOSE, R., 2002, "A Conceptual Framework for Composing and Managing Scientific Data Lineage". In: **Proceedings of International Conference on Scientific and Statistical Database Management**, Edinburgh, Scotland, pp.15-19, July.

BOZSAK, E. *et al.*, 2002, "KAON – Towards a Large Scale Semantic Web". In **Proceedings of Database and Expert Systems Applications (DEXA)**, Aix en Provence, France, September.

IBM BPWS4J. Available from: http://www.alphaworks.ibm.com/tech/bpws4j.

BRAGA, R., WERNER, C., MATTOSO, M., 1999, "Odyssey: A Reuse Environment based on Domain Models". In: **2nd IEEE Symposium on Application-Specific Systems and Software Engineering Technology** (ASSET'99), Richardson, USA, March.

BRAZ, M. H., MELO, R. N., 1989, “Modelagem e Gerência de Modelos em Sistemas de Apoio a’a Decisão”. In **Proceedings of XXII National Congress of Informatics**, pp. 197-202, São Paulo, SP, Brazil, September (in portuguese).

BUNEMAN, P., FOSTER, I., 2002, **Workshop on Data Provenance and Derivation**, Chicago, IL, USA, October.
Available from <http://people.cs.uchicago.edu/~yongzh/position_papers.html>

CARDOSO, L. F., SOUZA, J. M., MARQUES, C., 2002, "A Collaborative Approach to the Reuse of Scientific Experiments in the Bill of Experiments Tool". In: **The Seventh International Conference on Computer Supported Cooperative Work in Design**, pp. 296-301, Rio de Janeiro, Brazil, September.

CAREY, M. J., et al., 1995, **Towards Heterogeneous Multimedia Information Systems: The Garlic Approach**, Technical Report, IBM Almaden Research Center, USA.

CARNEIRO, F. L., 1996, **Análise Dimensional e Teoria da Semelhança e dos Modelos Físicos**. Ed. UFRJ, 2nd edition, Rio de Janeiro (in portuguese).

CARTIER, J., RUDOLPH, J., STEWART, J., 2001, **The Nature and Structure of Scientific Models**, Working Paper of NCISLA, Wisconsin Center for Education Research, University of Winconsin.

CAVALCANTI, M., MATTOSO, M., CAMPOS, M, LLIRBAT, F., SIMON, E., 2002, “Sharing Scientific Models in Environment Applications”. In: **Proceedings of ACM Symposium on Applied Computing**, pp. 453-457, Madrid, Spain, March.

CAVALCANTI, M., MATTOSO, M., CAMPOS, M. L., SIMON, E., LLIRBAT, F., 2002, “An Architecture for Managing Distributed Scientific Resources”. In: **Proceedings of IEEE International Conference on Scientific and Statistical Database Management**, Edinburgh, Scotland, pp.47-55, July.

CAVALCANTI, M., CAMPOS, M., MATTOSO, M., SANTOS, M., BARRETO, P., 2002, “Managing Scientific Models in Bio-phenomena Interpretation.”. In **Proceedings of World Petroleum Congress (WPC)**, Rio de Janeiro, Brazil, September.

CAVALCANTI, M., BAIAO, F., RÖSSLE, S., BISCH, P. M., TARGINO, R., PIRES, P. F., CAMPOS, M. L., MATTOSO, M., 2003, “Structural Genomic Workflows Supported by Web Services”. To appear in **Proceedings of Workshop on Biological Data Management**, CAiSE´03, Prague, September.

CAVALCANTI, M., CAMPOS, M. L., MATTOSO, M., 2002, “Managing Scientific Models in Structural Genomic Projects”. Position Paper at **Workshop on Data Provenance and Derivation**, Chicago, IL, USA, October.
Available from <http://people.cs.uchicago.edu/~yongzh/position_papers.html>

CAVALCANTI, M., MATTOSO, M., CAMPOS, M. L., 2003, “Managing Scientific Resources with Web Services”. To appear in **Proceedings of the Workshop on E-business and Semantic Web**, Austria, June.

CHANG, A., HOLSAPPLE, C., WHINSTON A., 1993, “Model Management Issues and Directions”, **Decision Support Systems**, vol.9, pp.19-37.

CHERVENAK, A., FOSTER, I., KESSELMAN, C., SALISBURY, C., TUECKE, S., 2001, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets”, **Journal of Network and Computer Applications**, 23, pp.187-200.

CHRISTOFOLETTI, A., 1999, “**Modelagem de Sistemas Ambientais**”. Ed. Edgard Blucher (in portuguese).

CHRISTOPHIDES, V., HOUSTIS, C., LALIS, S., TSALAPATA, H., 1999, “Ontology-driven Integration of Scientific Repositories”. In: **Proceedings of the 4th Workshop on Next Generation Information Technology and Systems** (NGITS’99), Zikhron-Yaakov, Israel, July.

CRITCHLOW, T., MUSICK, R., SLEZAK, T., 2001, “Experiences Applying Meta-Data to Bioinformatics”, **Information Sciences**, v.139 (1-2), Elsevier Science Inc., November.

CURBERA, F., GOLAND, Y, et al., 2002, **Business Process Execution Language for Web Services V1.0**, Microsoft, BEA and IBM.

CZAJKOWSKI, K., FOSTER, I., KARONIS, N., KESSELMAN, C., MARTIN, S., SMITH, W., TUECKE, S., 1998, "A Resource Management Architecture for Metacomputing Systems". In: **4thWorkshop on Job Scheduling Strategies for Parallel Processing**, Springer-Verlag, pp. 62-82.

DATAGRID PROJECT – Available from <http://www.gridcomputing.com/>.

DAVIS, J.B., 1967, **Petroleum Microbiology**, Elsevier Publishing Company, eds.Amsterdam, pp. 604.

DOLK, D., 1986, "A Generalized Model Management System for Mathematical Programming", **ACM Transactions on Mathematical Software**, vol. 12, No.2, pp. 92-126, June.

DOLK, D., KONSYNSKI, B. R., 1984, "Knowledge Representation for Model Management Systems", **IEEE Transactions on Software Engineering**, vol. SE-10(6), November.

DOMENIG, R., DITTRICH, K.R., 1999, "An Overview and Classification of Mediated Query Systems", **SIGMOD Record**, September.

DRIESSEN, P. J., DUDAL, R., 1991, "The major soils of the world", Agricultural University Wageningen.

DU, W., SHAN, M., 1996, 1996, "Query Processing in Pegasus". In: Bukhres, O., Elmagarmid, A. (eds), **Object-Oriented Multidatabase Systems: A Solution for Advanced Applications**, pp. 449-471, Prentice Hall.

EbXML registry.
Available from <http://www.ebxml.org/>

ECOBAS PROJECT
Available from <http://dino.wiz.uni-kassel.de/ecobas.html>

ECOBASE members, 2001, The Ecobase Project: Database and Web Technologies for Environmental Information Systems, **ACM Sigmod Record**, v.30 (3), pp.70-75, September.

ESP2Net Project – Available from <http://dml.cs.ucla.edu/projects/dml_esip>.

ESSW Project – Available from <http://essw.bren.ucsb.edu/>.

FEUVRIER, C. V., 1971, "**La Simulation des Systèmes**", Dunod, Paris.

FOSTER, I. AND KESSELMAN, C., 1999, "Globus: A Toolkit-based Grid Architecture". In: Foster, I., Kesselman, C. (eds), **The Grid: Blueprint for a New Computing Infrastructure**, Morgan Kauffman, pp. 259-278.

FOSTER I., KESSELMAN C, NICK J, TUECKE, S., 2002, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Working paper, January.
Available from <http://www.globus.org/research/papers/ogsa.pdf>

FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2003, "The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration". In: **Conference of Innovative Data Research – CIDR**, Asilomar, CA, USA, January.

FOSTER, I., KESSELMAN, C., 2001, "A Data Grid Reference Architecture", Technical Report, GriPhyN-2001-12.

FOSTER, I., KESSELMAN, C., NICK, J. M., TUECKE, S., 2002, "Grid Services for Distributed Systems Integration", **IEEE Computer**, pp.37-46, June.

FOSTER, I., VOECKLER, J., WILDE, M., ZHAO, Y., 2002, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation". In **International Conference on Scientific and Statistical Database Management**, Edinburgh, Scotland, pp.37-46, July.

FREW, J., BOSE, R., 2001, "Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products". In **Proceedings of International Conference Scientific Statistical DB Management**, Fairfax, VA, USA, pp.180-189, July.

GABELE, T., BENZ, J., HOCH, R., 1999, "Standardization of model documentation: Usage of ECOBAS model documentation system – a short introductory manual", **ECOMOD Newsletter**, ISEM, June.
Available from <http://ecomod.tamu.edu/ecomod/isem.html>.

GALHARDAS, H., SIMON, E., TOMASIC, A., 1998, "A Framework for Classifying Scientific Metadata", In **Proceedings of AAAI'98**.

GANNON, D., CHIU, K., *et al.*, 2002, **Analysis of the Open Grid Services Architecture**, Working paper,Department of Computer Science, Indiana University, Bloomington, IN, USA. Available from <http://www.extreme.indiana.edu/~gannon/OGSAanalysis3.html>

GARCIA, S., 2003, "Towards the Support of Domain-Ontology Construction", Doctor´s thesis Qualifying Exam, DCC/IM/NCE – UFRJ, to be presented.

GARCIA-MOLINA, H., PAPAKONSTANTINOU, Y., QUASS, D., et al., 1997, The TSIMMIS Approach to Mediation: Data Models and Languages, **Journal of Intelligent Information Systems**.
Available from <http://www-db.stanford.edu/tsimmis/publications.html>.

GARDARIN, G., GANNOUNI, S., FINANCE, B., 1996, "IRO-DB: A Distributed System Federating Object and Relational Databases". In: (eds.) O. Bukhres and A. Elmagarmid, **Object-Oriented Multidatabase Systems: A Solution for Advanced Applications**", pp. 684-712, Prentice Hall.

GEOFFRION, A. M., 1987, "An Introduction to Structured Modelling", **Management Science**, vol.33, no.5, pp. 547-588, May.

GEORGAKOPOULOS, D., HORNICK, M., SHETH, A., 1995, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", **Distributed and Parallel Databases**, Vol. 3, pp. 119-153

GILLMANN, M., WEISSENFELS, J., WEIKUM, G., KRAISS, A., 2000, "Performance and Availability Assessment of the Configuration of Distributed Workflow Management Systems". In: **Proceedings of 7th International Conference on Extending Database Technology**, EDBT'2000, Konstanz, Germany, March.

GLEISER, M., 2002, "O 'porquê' e o 'como'", **Folha de São Paulo Newspaper**, São Paulo, June (in portuguese).

GLOBUS – Available from <http://www.globus.org>.

GOA SYSTEM – Available from <http://www.cos.ufrj.br/~goa>.

GOLDBARG, M., LUNA, H., 2000, **Otimização Combinatória e Programação Linear: Modelos e Algoritmos**, Ed. Campus.

GONÇALVES, F., 2003, "**Supporting Data Integration in Mediation Architectures**", M.Sc. Dissertation, COPPE Sistemas, UFRJ, to be presented.

GRANT, W. E., 1986, **Systems Analysis and Simulation in Wildlife and Fisheries Sciences**, New York, John Wiley & Sons.

GRAY, J., 2002, "**Mining the Sky – The World Wide Telescope",** Invited talk at the Workshop on Distributed Data and Structures – WDAS-2002, University Paris 9, March.
Available from <http://www.research.microsoft.com/~Gray/talks>.

Grid Computing Info Centre – Available from <http://www.gridcomputing.com/>.

Grid Datafarm Project – Available from <http://datafarm.apgrid.org/overview.en.html>.

GriPhyN Project – Available from <http://www.griphyn.org>

GUARINO, N., GIARETTA, P., 1995, "Ontologies and Knowledge Bases: Towards a Terminological Clarification". In: N. Mars (ed), **Torwards Very Large Knowledge Bases: Knowledge Building a Knowledge Sharing,** IOS Press, Amsterdam, pp.25-32.

GUARISO, G., HITZ, M., WERTHNER, H., 1996, "An Integrated Simulation And Optimization Modelling Environment For Decision Support", **Decision Support Systems**, Vol.16, pp. 103-117.

GUENTHER, O., VOISARD, A., 1997, "Metadata in Geographic and Envrionmental Data Management". In: W. Klas and A. Sheth, editors, **Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data** McGraw Hill.

GUPTA, P., LIN, E. T., 1994, "DataJoiner: A Practical Approach to Multi-Database Access". In: **Proceedings of PDIS 1994**, pp. 264.

HAEFNER, J. W., 1996, "**Modeling Biological Systems: Principles and Applications**", London, Chapman & Hall.

HAGGET, P., CHORLEY, R. J., 1967, "Models, Paradigms and New Geography", in **Models in Geography**, London, Methuen & Co. *Apud* CHRISTOFOLETTI, A., 1999

HOLLINGSWORTH, D., 1995, "**The Workflow Reference Model**", Workflow Management Coalition, Document TC00-1003, Issue 1.1, January.

HORROCKS I., 2002, "DAML+OIL: a reason-able Web ontology language". In **Proceedings of EDBT 2002**, March.

HOUSTIS, C., LALIS, S., 2001, "ARION: A Scalable Architecture for a Digital Library of Scientific Collections". In: **8th Panhellenic Conference on Information**, November.

HOUSTIS, C., LALIS, S., CHRISTOPHIDES, V., PLEXOUSAKIS, D., VAVALIS, M., PITIKAKIS, M., KRITIKOS, K., SMARDAS, A., GIKAS, X., 2002, "A Service Infrastructure for e-Science: the case of the ARION system". In **Workshop on Web Services, e-business, and the Semantic Web**, Toronto, Ontario, Canada, May.

HOUSTIS, C., NIKOLAOU, C., LALIS, S., KAPIDAKIS, S., CHRISTOPHIDES, V., SIMON, E., TOMASIC, A., 1999, "Towards a Next Generation of Open Scientific Data Repositories and Services", **CWI Quarterly**, Vol. 12, No.12, Special Issue on Digital Libraries, Amsterdam, June.

HSU, M., KLEISSNER, C., 1996, "ObjectFlow: Towards a Process Management Infrastructure", In **Distributed and Parallel Databases**, vol. 4, pp. 169-194**.**

HULL, R., LLIRBAT, F., SIMON, E., SU, J., DONG, G., KUMAR, B., ZHOU, G., 1999, "Declarative Workflows that Support Easy Modification and Dynamic Browsing". In: **Proceedings of ACM International Joint Conference on Work Activities Coordination, WACC'99**, February, San Francisco, CA, USA, pp.69-78.

INMON, B., 1996, **Building the Data Warehouse**, John Wiley & Sons, Inc.

IOANNIDIS, Y., LIVNY, M., GUPTA, S., PONNEKANTI, N., 1996, "ZOO: A Desktop Experiment Management Environment". In: **Proceedings of the 22nd VLDB Conference**, pp. 274-285, Bombay, India.

IVERSON, L., PRASAD, A., SCHWARTZ, M., 1999, "Modeling potential future individual tree-species distributions in the eastern United States under a climate change scenario: a case study with Pinus virginiana", **Ecological Modeling**, No. 115, pp. 77-93.

KAESTLE, G., SHEK, E.C. AND DAO, S. K., 1999, "Sharing Experiences from Scientific Experiments". In: **Proceedings Int. Conf. On Statistical and Scientific Database Management**, pp.168-177, July.

KANEHISA, M., 2000, **Post-Genome Informatics**, Oxford University Press.

KIM, W., CHOI, I., GALA, S., SCHEEVEL, M., 1993, "On Resolving Schematic Heterogeneity", **International Journal of Parallel Distributed Databases**", vol. 1, pp. 251-279.

KITCHENHAM, B., PICKARD, L., PFLEEGER, S., 1995, "Case Studies for Method and Tool Evaluation", **IEEE Software**, pp.52-62, July.

KOBRYN, C., 1999, "UML 2001: A Standardization Odyssey", **ACM Communications**, Vol. 42, No. 10, October.

KUZNETSOVA, V.A., 1960, "Occurrence of sulfate-reducing organisms in oil-bearing formations of Kuibyshev region with reference to salt composition of layer waters", **Mikrobiologiya**, 29, pp. 408-414.

LASKOWSKI, R. A., MACARTHUR, M. W., MOSS, D. S., THORNTON, J. M., 1993, "PROCHECK: a program to check the stereochemical quality of protein structures", **Journal of Appl. Cryst.,** 26, pp. 283-291.

LEAL, L. N., 2003, "Gerador de Aplicações para Repositórios de Metadados baseado em XSL e XML Schema", **Final Project Monography,** Dept.Computer Science, DCC-IM, UFRJ, March.

LEE, B., SNAPP, R., MUSICK, R., CRITCHLOW, T., 2001, "Ad hoc Query Support for Very Large Simulation Mesh Data: the Metadata Approach". In: **Proceedings of Brazilian Symposium on**

Databases, pp.199-212, Rio de Janeiro, Brazil, October.

LENARD, M. L., 1993, "An Object-oriented approach to model management", **Decision Support Systems,** vol. 9, pp. 67-73.

LE SELECT System, Caravel Project, INRIA, France.
Available from <http://www-caravel.inria.fr/LeSelect/>.

LESK, M., 2002, "Data Provenance and Derivation", Position paper at **Workshop on Data Provenance and Derivation**, Chicago, IL, USA, October.
Available from <http://people.cs.uchicago.edu/~yongzh/position_papers.html>

LEYMANN, F. AND ALTERHUBER, W., 1994, "Managing Business Processes as an Information Resource", **IBM Systems Journal**, 33, pp. 346-347.

LIE, H. W., SAARELA, J., 1999, "Multipurpose Web Publishing: Using HTML, XML, and CSS", **Communications of the ACM**, Vol. 42, No. 10, October.

LIU, L., PU, C., LEE, Y., 1996, "An adaptive approach to query mediation across heterogeneous databases". In: **Proceedings of the Int. Conf. Cooperative Information Systems**, IEEE Press, pp.144-156, June.

LIU, L., PU, C., LEE, Y., 1995, "**The DIOM approach to Large-scale Interoperable Database Systems**", Technichal Report TR95-16, Department of Computing Science, University of Alberta, March.

LJUNG, L., 1987, "**System Identification Theory for the User**", Englewood Cliffs, New Jersey Prentice Hall.

LONG, D., MANTEY, P., WITTENBRINK, C., HAINING, T., MONTAGUE, B., 1995, "REINAS: the Real-Time Environmental Information Network and Analysis System". In: **Proceedings of Technologies for the Information Highway**, COMPCON'1995, pp. 482-487, San Francisco, CA, USA.

MATTOSO, M., CAVALCANTI, M. C., PINHEIRO, R., VIEIRA, H., AZEVEDO, L., MARQUES, C., MONTEIRO, R., GONÇALVES, F., WERNER, C. 2002, "Gerência de Documentos XML no GOA". In: **Brazilian Symposium of Software Engineering**, Tools Section**,** Gramado, RS, Brazil, October.

MAYER, M. K., 1998, "Future trends in model management systems: parallel and distributed extensions", **Decision Support Systems**, Vol 22, pp. 325-335.

MDC (Metadata Coalition), 1999, **Open Information Model**, version 1.1, August.

MEDEIROS, C**.,** VOSSEN, G., WESKE, M., 1995, "WASA: A Workflow-Based Architecture to Support Scientific Database Applications". In: **International Workshop and Conference on Database and Expert Systems Applications** (DEXA), pp. 574–583 London, U.K., September.

MEYER, L. A., 2003, "Grid Systems Support to Biologic Workflows", D.Sc. Thesis Qualifying Exam, COPPE Sistemas – UFRJ, to be presented.

Object Management Group, 1997, "**Meta-Object Facility**", OMG TC document cf/97-01-01, Linnæus Project, DSTC, January.
Available from <http://www.omg.org>

MOURA, F.A., 2001, "Suporte à Decisão em Interpretação de Fenômenos", M.Sc. Thesis, DCC/NCE - UFRJ, March.

Microsoft C/C++ Language Reference. Available from <http://msdn.microsoft.com>

MUETZELFELDT, R. AND TAYLOR, J., 2001, "Developing forest models in the Simile visual modelling environment". In: **IUFRO 4.11 Conference on Forest biometry, modelling, and information science**, Greenwich (UK), June.

MyGrid Project. Available from <http://www.mygrid.org.uk>.

NCBI BLAST – Available from <http://www.ncbi.nlm.nih.gov/blast/Blast.cgi>.

NIEMCZYK, B., 2002, MySQL-XML Library v1.2.
Available from <www.sourceforge.com>

NOY, N.F., FERGERSON, R. W., MUSEN, M. A., 2000,"The knowledge model of Protege-2000: Combining interoperability and flexibility", In: **2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)**, Juan-les-Pins, France.

OGSA-DAI project.
Available from <http://www.ogsadai.org.uk>.

OGSI Workgroup, 2003, "Open Grid Service Infrastructure", Draft document, Global Grid Forum, February.
Available from <http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf>

Object Management Group, 1991, **The Common Object Request Broker: Architectures and Specification,** Framingham, Massachusetts.

Object Management Group – "**Common Warehouse Metamodel**" – OMG specification document of CWM, v1.0
Available from http://www.omg.org/technology/cwm/

PATRIKALAKIS, N., ABRAMS, S., BELLINGHAM, J., CHO, W., MIHANETZIS, K., ROBINSON, A., SCHMIDT, H., WARIYAPOLA, P., 2000, "The Digital Ocean". Invited paper in **Proceedings of Computer Graphics International, GCI'2000**, pp. 45-53, Geneva, Switzerland, IEEE Computer Society Press. Los Alamitos, CA, June.

PINTO, G. R. B., STRAUCH, J. C. M., CARDOSO, L. F., et al., 2002, "A Framework to Support Scientific Knowledge Management: a Case Study in Agro-meteorology". In: **The Seventh International Conference on CSCW in Design**, pp. 320-324, Rio de Janeiro, Brasil, September.

PIRES, P., 1997, "**HIMPAR, Uma Arquitetura para Interoperabilidade de objetos Distribuídos**", M.Sc. Thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

DECAIR Project, EC research project, 1999-2002.
Available from http://www-rocq.inria.fr/air/decair/

Registry of Ecological Models (REM) – Available from http://dino.wiz.uni-kassel.de/ecobas.html.

RICHARDSON, G. P., PUGH III, A. L., 1999, "Introduction to Systems Dynamics Modeling", **Systems Dynamics Series**, PEGASUS Communications, Inc., Waltham, MA, USA.

RICHMOND, B., 1984, "System Dynamics/Systems Thinking: Let's Just Get On With It". In: **International Systems Dynamics Conference**, Sterling, Scotland.

RIZZOLI, A. E., DAVIS, J. R., ABEL, D. J., 1998, "Model and data integration and re-use in environmental decision support systems", **Decision Support Systems**, Vol. 24, pp. 127-144.

RODRIGUEZ, M., ROUSSOPOULOS, N., 2000, "Automatic Deployment of Application-Specific Metadata and Code in MOCHA", to appear in **Proceedings of EDBT 2000.**

ROSETTANET. Available from <http://www.rosettanet.org>

RÖSSLE, S., RIBEIRO, S., et al., 2002, "**A Computational Environment Development for the Application of Homology Modeling Methods in Structural Genomic Projects**", Project Poster, IBCCF, UFRJ, Brazil (in portuguese).

RUDIO, F. V., 1978, **Introdução ao Projeto de Pesquisa Científica**, 29th edition, Editora Vozes, Petrópolis, RJ, Brazil (in portuguese).

ŠALI, A., 2001, **MODELLER: A Program for Protein Structure Modeling Release 6**, Rockefeller University.

SCHÖPP, W., AMANN, M., COFALA, J., HEYES, C., KLIMONT, Z., 1999, "Integrated assessment of European aire pollution emission control strategies", **Environmental Modelling & Software**, No. 14, pp. 1-9.

SCOTT, E. M., 1996, "Uncertainty and Sensitivity Studies of Models of Environmental Systems". In: **Proceedings of Winter Simulation Conference**, (eds.) J. Charnes, D. Morrice, D. Brunner and J. Swain, San Diego, CA, USA, December.

SHETH, A. P., LARSON, J. A., 1990, "Federated Database System for Managing Distributed, Heterogeneous, and Autonomous Databases". In: **ACM Computing Surveys**, v. 22(3), September.

SHETH, A., GEORGAKOPOULOS, D., JOOSTEN, S., RUSINKIEWICZ, M., SCACCHI, J., WOLF, A., 1996, "**Report from the NSF Workshop on Workflow and Process Automation in Information Systems",** Technical Report, Large-Scale Distributed Information Systems Lab, University of Georgia, GA, USA, October.

SINGH, M.P., VOUK, M.A., 1996, "Scientific workflows: scientific computing meets transactional workflows". In: **Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions**, Univ. Georgia, Athens, GA, USA, pp. 28-34.

SINGH, V. P., 1995, **Computer Models of Watershed Hydrology**, Boulder, Water Resources Publications.

SPRIET, J. A., VANSTEENKISTE, G. C., 1982, **Computer-aided modelling and simulation**, Dept. of Applied Mathematics and Biometrics, University of Ghent, Belgium, Academic Press, Inc..

SPYNS, P, MEERSMAN, R., JARRAR, M., 2002, "Data Modelling versus Ontology Engineering", **SIGMOD Record** 31(4), pp.12-17.

TARGINO, R., 2003, "Workflows Científicos apoiados por Web services", M.Sc. Dissertation, to be presented, COPPE Sistemas, UFRJ, Brazil.

TEIXEIRA, F., 2003 "Fábrica de Web services para Dados Científicos", M.Sc. Dissertation, to be presented, COPPE Sistemas, UFRJ, Brazil.

TEMPLETON, M. *et al*., 1987, "Mermaid – A Front End to Distributed Heterogeneous Databases". In: **Proceedings of IEEE**, vol.75, no. 75, pp. 695-708, May.

TOMASIC, A., RACHID, L., VALDURIEZ, P., 1998, "A Data Model and Query Processing Techniques for Scaling Access to distributed Heterogeneous Databases in Disco". In: **IEEE Transactions on Knowledge and Data Engineering,** v.10 (4), July.

TOMASIC, A., SIMON, E., 1998, "Improving Access to Environmental Data using Context Information", **SIGMOD Record**, January.

APACHE, TOMCAT. Available from http://jakarta.apache.org.

UCHÔA, E. M. A., MELO, R. N., 1999, "Integração de Sistemas de Bancos de Dados Heterogêneos Usando Frameworks". In **Proceedings of 14th Database Brazilian Symposium (SBBD'99)**, pp.381-393, Florianópolis, SC, Brazil.

UCHÔA, E.M.A., LIFSCHITZ, S., MELO, R.N., 1998, "HEROS: a Heterogeneous Object Oriented Database System". In **9th International Conference and Workshop on Database and Expert Systems Applications (DEXA'98)**.

UDDI Executive White Paper. Available from <http://www.uddi.org/>

UML Revision Task Force, 1999, **OMG Unified Modeling Language Specification**, v.1.3, document ad//990608, Object Management Group, June.

UNSPSC. Available from <http://www.unspsc.org>

USHOLD, M., KING, M., 1995, "Towards a Methodology for Building Ontologies". In **Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing** (UCAI95).

VAN DER AALST, W., BARROS, A., HOFSTEDE, A., KIEPUSZEWSKI, B., 2000, "Advanced Workflow Patterns". In: **Conference on Cooperative Information Systems**, pp. 18-29.

VASUDEVAN, V., 1999, **Notes on ontologies**, Technical Note, Object Services and Consulting, Inc., February.
Available from <http://www.objs.com/agility/tech-reports/9902-ontology.html>

VENKATARAMAN, A., ZHANG, T., 1998, "Heterogeneous Database Query Optimization in DB2 Universal DataJoiner". In: **Proceedings of VLDB 1998**, pp. 685-689.

VIEIRA, H. RUBERG, G. MATTOSO, M., 2002, "*Xverter:* Armazenamento e Consulta de Dados XML em SGBDs". In: **Proceedings of the Brazilian Symposium on Databases (SBBD)**, Gramado, RS, Brazil, October.

WEINSTEIN, P., 1998, "Ontology-Based Metadata: Transforming the MARC Legacy". In: **Proceedings of the 3rd ACM International Conference on Digital Libraries**, Pittsburgh, PA USA, pp. 254-263, June.

WESKE, M., VOSSEN, G., MEDEIROS, C**.,** 1996, **Scientific Workflow Management: WASA Architecture and Applications**, Schriften zur Angewandten Mathematik und Informatik 03/96-I, Universität Münster.

WfMC Work Group I, 1999, **Interface I: Process Definition Interchange Process Model**, WfMC TC-1016-P, Version 1.1, October.

WfMC, 1999, **WfMC Terminology and Glossary**, WfMC TC-1011, Issue 3.0, February.

WIDENIUS, M., AXMARK, D., 2002, "**MySQL Reference Manual**", O'Reilly and Associates, June.

WIEDERHOLD, G., 1992, "Mediators in the architecture of future information systems". In **IEEE Computer**, v.25, pp. 38-49.

WIEDERHOLD, G., 1994, "An Ontology Algebra". In: **Proceedings of the Monterey Workshop on Formal Methods**, Luqi (ed.), U.S. Naval Post Graduate School, September.

WIEDERHOLD, G., 1994, "Interoperation, Mediators and Ontologies" In: **Proceedings International Symposium on Fifth Generation Computer Systems (FGCSOB94)**, Workshop on Heterogeneous Cooperative Knowledge-Bases, Vol.W3, pp. 33-48, ICOT, Tokyo, Japan, December.

WOLDENBERG, M. J., 1985, **Models in Geomorphology**, London, Allen & Unwin.

WROE, C., STEVENS, R., GOBLE, C., ROBERTS, A., GREENWOOD, M., 2003, "A suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data". In **International Journal of Cooperative Information Systems**, in press.

W3C Web Services – Available from <http://www.w3c.org/Webservices>.

Web Services Architecture, W3C Working Draft, November, 2002.
Available from <http://www.w3.org/TR/2002/WD-ws-arch-20021114/>

WSDL Schema – Available from <http://schemas.xmlsoap.org/wsdl/>

W3C Web Service Description Language Working Draft, version 1.2., January, 2003.
Available from <http://www.w3.org/TR/wsdl12/>

WSFL v 1.0, Web Services Flow Language, IBM.
Available from <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf >

XERCES, DOM API implementation, Apache Project.
Available from <http://jakarta.apache.org>

XHUMARI, F. *et al.*, 2000, "Le Select User Manual", INRIA.
Available from <http://www-caravel.inria.fr/~leselect/doc/UserManual.rtf.gz>

XML Query – Available from <http://www.w3.org/XML/Query>.

XML Schema – Available from <http://www.w3.org/XML/Schema>.

XSL – Available from <http://www.w3.org/Style/XSL/>.

9. Appendix

9.1 SPMW XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.nce.ufrj.br/yoko/spm"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:spm="http://www.nce.ufrj.br/yoko/spm"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:annotation>
        <xsd:documentation>
         Scientific Resources Description  MetaSchema
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="ScientificResourceDefinitions">
        <xsd:complexType>
            <xsd:group ref="spm:gResource" maxOccurs="unbounded"/>
        </xsd:complexType>
    </xsd:element>
    <!-- At least one of the elements below should be published, and there is no forced order -->
    <xsd:group name="gResource">
        <xsd:choice>
            <xsd:element ref="spm:ModDC"/>
            <xsd:element ref="spm:ProgDC"/>
            <xsd:element ref="spm:Model"/>
            <xsd:element ref="spm:Program"/>
            <xsd:element ref="spm:DataResource"/>
            <xsd:element ref="spm:CodeResource"/>
            <xsd:element ref="spm:Workflow "/>
            <xsd:element ref="spm:Experiment"/>
        </xsd:choice>
    </xsd:group>
    <!-- Declaração dos elementos principais e suas chaves primárias e estrangeiras -->
    <xsd:element name="ModDC" type="spm:tModDC">
        <xsd:key name="kModDC">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idDC"/>
        </xsd:key>
    </xsd:element>
    <xsd:element name="ProgDC" type="spm:tProgDC">
        <xsd:key name="kProgDC">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idDC"/>
        </xsd:key>
        <xsd:keyref name="refModDC" refer="spm:kModDC">
            <xsd:selector xpath="."/>
            <xsd:field xpath="implements"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="ModAttribute" type="spm:tMDCAttribute">
        <xsd:key name="kModAttribute">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idAttribute"/>
        </xsd:key>
    </xsd:element>
    <xsd:element name="ProgAttribute" type="spm:tPDCAttribute">
        <xsd:key name="kProgAttribute">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idAttribute"/>
        </xsd:key>
        <xsd:keyref name="refModAttribute" refer="spm:kModAttribute">
            <xsd:selector xpath="."/>
            <xsd:field xpath="implements"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="Model" type="spm:tModel">
```

```
        <xsd:key  name="kModel">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idTF"/>
        </xsd:key>
        <xsd:keyref name="refInputModDC" refer="spm:kModDC">
            <xsd:selector xpath="input"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
        <xsd:keyref name="refOutputModDC" refer="spm:kModDC">
            <xsd:selector xpath="output"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
        <xsd:keyref name="refModParm" refer="spm:kModDC">
            <xsd:selector xpath="parm"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="Program" type="spm:tProgram">
        <xsd:key  name="kProgram">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idTF"/>
        </xsd:key>
        <xsd:keyref name="refModel" refer="spm:kModel">
            <xsd:selector xpath="."/>
            <xsd:field xpath="implements"/>
        </xsd:keyref>
        <xsd:keyref name="refInputProgDC" refer="spm:kProgDC">
            <xsd:selector xpath="input"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
        <xsd:keyref name="refOutputProgDC" refer="spm:kProgDC">
            <xsd:selector xpath="output"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
        <xsd:keyref name="refProgParm" refer="spm:kProgDC">
            <xsd:selector xpath="parm"/>
            <xsd:field xpath="refersTo"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="DataResource" type="spm:tDataResource">
        <xsd:key  name="kDataResource">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idDR"/>
        </xsd:key>
        <xsd:keyref name="refDescribedByProgDC" refer="spm:kProgDC">
            <xsd:selector xpath="."/>
            <xsd:field xpath="spm:describedBy"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="CodeResource" type="spm:tCodeResource">
        <xsd:key  name="kCodeResource">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idCR"/>
        </xsd:key>
        <xsd:keyref name="refDescribedByProgram" refer="spm:kProgram">
            <xsd:selector xpath="."/>
            <xsd:field xpath="spm:describedBy"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="Workflow" type="spm:tWorkflow">
        <xsd:key  name="kWorkflow">
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idWF"/>
        </xsd:key>
        <xsd:keyref name="refWfStep" refer="spm:kProgram">
            <xsd:selector xpath="."/>
            <xsd:field xpath="spm:wfStep"/>
        </xsd:keyref>
    </xsd:element>
    <xsd:element name="Experiment" type="spm:tExperiment">
        <xsd:key  name="kExperiment">
```

```
            <xsd:selector xpath="."/>
            <xsd:field xpath="@idEx"/>
        </xsd:key>
        <xsd:keyref name="refWorkflow" refer="spm:kWorkflow">
            <xsd:selector xpath="."/>
            <xsd:field xpath="spm:workflow "/>
        </xsd:keyref>
        <xsd:keyref name="refInstanceOf" refer="spm:kWorkflow">
            <xsd:selector xpath="spm:essay"/>
            <xsd:field xpath="spm:instanceOf"/>
        </xsd:keyref>
        <xsd:keyref name="refCodeResource" refer="spm:kCodeResource">
            <xsd:selector xpath="spm:essay/spm:execution"/>
            <xsd:field xpath="spm:codeResource"/>
        </xsd:keyref>
        <xsd:keyref name="refDataResource" refer="spm:kDataResource">
            <xsd:selector xpath="spm:essay/spm:execution/spm:dataMatch"/>
            <xsd:field xpath="spm:dataResource"/>
        </xsd:keyref>
    </xsd:element>
    <!-- Declarando tipos complexos -->
    <xsd:complexType name="tDataCategory" abstract="true">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="creator" type="xsd:string"/>
            <xsd:element name="creationDate" type="xsd:date"/>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>to support extensibility elements </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
        <xsd:attribute name="idDC" type="xsd:NCName" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="tModDC">
        <xsd:complexContent>
            <xsd:extension base="spm:tDataCategory">
                <xsd:sequence>
                    <xsd:element name="MDCAttribute" maxOccurs="unbounded">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="attItem" type="spm:tMDCAttribute"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tProgDC">
        <xsd:complexContent>
            <xsd:extension base="spm:tDataCategory">
                <xsd:sequence>
                    <xsd:element name="implements" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="wsdlElementRef" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation> Each ProgDC corresponds to a type in a wsdl
document.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="PDCAttribute" maxOccurs="unbounded">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="attItem" type="spm:tPDCAttribute"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tAttribute" abstract="true">
        <xsd:sequence>
            <xsd:element name="attTitle" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="idAttribute" type="xsd:NCName" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="tMDCAttribute">
        <xsd:complexContent>
            <xsd:extension base="spm:tAttribute">
                <xsd:sequence>
                    <xsd:element name="quantity" type="spm:tQuantity"/>
                    <xsd:element name="classification" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tPDCAttribute">
        <xsd:complexContent>
            <xsd:extension base="spm:tAttribute">
                <xsd:sequence>
                    <xsd:element name="unit" type="spm:tUnit"/>
                    <xsd:element name="format" type="xsd:string"/>
                    <xsd:element name="implements" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="wsdlElementRef" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation> Each PDCAttr corresponds to a type in a wsdl
document.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tTransformation" abstract="true">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="creator" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation>name of the creator of the transformation (not the person who
describes it)</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="creationDate" type="xsd:date">
                <xsd:annotation>
                    <xsd:documentation>date of the creation (conception) of the transformation. It should reflect
the model age.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="input" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="inputItem" type="spm:tDataIO"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="parm" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="parmItem" type="spm:tParm"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="output" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="outputItem" type="spm:tDataIO"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="constraint" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
```

```
                    <xsd:sequence>
                        <xsd:element name="constItem" type="spm:tConstraint"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>to support extensibility elements </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
        <xsd:attribute name="idTF" type="xsd:NCName" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="tModel">
        <xsd:complexContent>
            <xsd:extension base="spm:tTransformation">
                <xsd:sequence>
                    <xsd:element name="area" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>A model is usually associated to an area of application. Ex.::
industrial, economic, social, political, environmental, etc</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="scope" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>The target or scope of a model is the system it represents. Ex.:
Itajaí hydrographic basin, a geographic region, or an enterprise.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="classification" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>There are many different ways of classifying a model. Ex.
mathematic, logic, deductive, empiric, probabilistic, algorithmic, simulation, etc.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="purpose" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>Each model has a specific purpose, for which it is
valid.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="hypothesis" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>Every model is initially a hypothesis. Building a model
represents the expression of a scientific hypothesis that needs to be validated </xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="bibliographicRef" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation> scientific publications and related explanatory
material</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="webReference" type="xsd:string" minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation>web address of the model reference material, if
exists</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tProgram">
        <xsd:complexContent>
            <xsd:extension base="spm:tTransformation">
                <xsd:sequence>
                    <xsd:element name="implementationLanguage" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>Programming language with which the program was
implemented. It might be important to specify the version of the language.</xsd:documentation>
```

```
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="version" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>version/release of the program. </xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="implements" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="wsdlElementRef" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation> Each Program corresponds to a port type operation in a wsdl
document.</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tDataIO">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="refersTo" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="tParm">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="refersTo" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="tConstraint">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="description" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation>description in natural language of the constraint</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="expression" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation>description in a formal  language of the constraint (possibly
BPEL4WS)</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="tResource" abstract="true">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="creator" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation>if the creator is a program, it should be identified by a uri, a program
name, or a code execution id</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="creationDate" type="xsd:date"/>
            <xsd:element name="describedBy" type="xsd:string"/>
            <xsd:element name="wsdlElementRef" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation> Each code/data resource corresponds to a port operation in a wsdl
document.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>to support extensibility elements </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
        <xsd:attribute name="idResource" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="tDataResource">
        <xsd:complexContent>
            <xsd:extension base="spm:tResource">
                <xsd:sequence>
                    <xsd:element name="provenance" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation>identification of data provenance. Ex.: name of the satellite,
sensor identification</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="genMechanism" type="spm:tGenMechanism">
                        <xsd:annotation>
                            <xsd:documentation>generation mechanism that was used to  generate data. Ex.:
satellite, sensor, code execution</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="webReference" type="xsd:string" minOccurs="0">
                        <xsd:annotation>
                            <xsd:documentation> If there is a web address that directly points to the Data
Resource, as an XML page</xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tCodeResource">
        <xsd:complexContent>
            <xsd:extension base="spm:tResource">
                <xsd:sequence>
                    <xsd:element name="operationalSystem" type="xsd:string"/>
                    <xsd:element name="hardwareInfo" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tWorkflow">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="creator" type="xsd:string"/>
            <xsd:element name="creationDate" type="xsd:string"/>
            <xsd:element name="wfDefinition" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation>address of the workflow specification using
BPEL4WS.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="wfStep" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>to support extensibility elements </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
        <xsd:attribute name="idWF" type="xsd:NCName" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="tExperiment">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="creator" type="xsd:string"/>
            <xsd:element name="creationDate" type="xsd:date"/>
            <xsd:element name="project" type="xsd:string"/>
            <xsd:element name="purpose" type="xsd:string"/>
            <xsd:element name="hypothesis" type="xsd:string"/>
            <xsd:element name="report" type="xsd:string"/>
            <xsd:element name="status" type="spm:tStatus"/>
            <xsd:element name="workflow" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="essay" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="creationDate" type="xsd:date"/>
                        <xsd:element name="creationTime" type="xsd:time"/>
                        <xsd:element name="comment" type="xsd:time" minOccurs="0">
                            <xsd:annotation>
                                <xsd:documentation>Each essay may generate some comments added by the
scientist.</xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                        <xsd:element name="instanceOf" type="xsd:string"/>
                        <xsd:element name="concreteWFdefinition" type="xsd:string"/>
                        <xsd:element name="duration" type="xsd:float"/>
                        <xsd:element name="execution" maxOccurs="unbounded">
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="codeResource" type="xsd:string"/>
                                    <xsd:element name="dataMatch" maxOccurs="unbounded">
                                        <xsd:complexType>
                                            <xsd:sequence>
                                                <xsd:element name="dataIO" type="xsd:string"/>
                                                <xsd:element name="dataResource" type="xsd:string"/>
                                            </xsd:sequence>
                                        </xsd:complexType>
                                    </xsd:element>
                                    <xsd:element name="parmMatch" minOccurs="0"
maxOccurs="unbounded">
                                        <xsd:complexType>
                                            <xsd:sequence>
                                                <xsd:element name="parm" type="xsd:string"/>
                                                <xsd:element name="dataResource" type="xsd:string"/>
                                            </xsd:sequence>
                                        </xsd:complexType>
                                    </xsd:element>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>to support extensibility elements </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:sequence>
        <xsd:attribute name="idEx" type="xsd:NCName" use="required"/>
    </xsd:complexType>
    <xsd:simpleType name="tGenMechanism">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="satellite"/>
            <xsd:enumeration value="sensor"/>
            <xsd:enumeration value="manual"/>
            <xsd:enumeration value="code execution"/>
            <xsd:enumeration value="other"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="tStatus">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="active"/>
            <xsd:enumeration value="notStarted"/>
            <xsd:enumeration value="interrupted"/>
            <xsd:enumeration value="suspended"/>
            <xsd:enumeration value="finished"/>
            <xsd:enumeration value="archived"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="tUnit">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="--- no unit ---"/>
            <xsd:enumeration value="ms:Shot"/>
            <xsd:enumeration value="ms:CarMile"/>
            <xsd:enumeration value="ms:FixedRate"/>
            <xsd:enumeration value="ms:GramsperCubicCentimeter"/>
```

```
<xsd:enumeration value="ms:KilogramsperSquareMeter"/>
<xsd:enumeration value="ms:Millivolts"/>
<xsd:enumeration value="ms:Manmonth"/>
<xsd:enumeration value="ms:SuperBulkBag"/>
<xsd:enumeration value="ms:FiftyPoundBag"/>
<xsd:enumeration value="ms:Milliamperes"/>
<xsd:enumeration value="ms:Megabyte"/>
<xsd:enumeration value="ms:PartsPerMillion"/>
<xsd:enumeration value="ms:Ampere"/>
<xsd:enumeration value="ms:Volt"/>
<xsd:enumeration value="ms:KiloPoundsPerSquareInch"/>
<xsd:enumeration value="ms:FootPounds"/>
<xsd:enumeration value="ms:Joules"/>
<xsd:enumeration value="ms:TenKilogramDrum"/>
<xsd:enumeration value="ms:Acre"/>
<xsd:enumeration value="ms:Atmosphere"/>
<xsd:enumeration value="ms:Billet"/>
<xsd:enumeration value="ms:Bale"/>
<xsd:enumeration value="ms:BaseBox"/>
<xsd:enumeration value="ms:Bucket"/>
<xsd:enumeration value="ms:Bundle"/>
<xsd:enumeration value="ms:Beam"/>
<xsd:enumeration value="ms:BoardFeet"/>
<xsd:enumeration value="ms:Bag"/>
<xsd:enumeration value="ms:Bar"/>
<xsd:enumeration value="ms:Block"/>
<xsd:enumeration value="ms:Bulk"/>
<xsd:enumeration value="ms:Bottle"/>
<xsd:enumeration value="ms:Barrel"/>
<xsd:enumeration value="ms:Bushel"/>
<xsd:enumeration value="ms:Box"/>
<xsd:enumeration value="ms:MillionBTUs"/>
<xsd:enumeration value="ms:Centipoise"/>
<xsd:enumeration value="ms:Case"/>
<xsd:enumeration value="ms:Carboy"/>
<xsd:enumeration value="ms:CubicCentimeter"/>
<xsd:enumeration value="ms:Carat"/>
<xsd:enumeration value="ms:CubicFeet"/>
<xsd:enumeration value="ms:Container"/>
<xsd:enumeration value="ms:CubicInches"/>
<xsd:enumeration value="ms:Cylinder"/>
<xsd:enumeration value="ms:Centimeter"/>
<xsd:enumeration value="ms:Can"/>
<xsd:enumeration value="ms:Crate"/>
<xsd:enumeration value="ms:Cartridge"/>
<xsd:enumeration value="ms:CubicMeter"/>
<xsd:enumeration value="ms:Cassette"/>
<xsd:enumeration value="ms:Carton"/>
<xsd:enumeration value="ms:Cup"/>
<xsd:enumeration value="ms:HundredPounds"/>
<xsd:enumeration value="ms:Coil"/>
<xsd:enumeration value="ms:CubicYard"/>
<xsd:enumeration value="ms:Days"/>
<xsd:enumeration value="ms:Degree"/>
<xsd:enumeration value="ms:Dram"/>
<xsd:enumeration value="ms:Miles"/>
<xsd:enumeration value="ms:Decimeter"/>
<xsd:enumeration value="ms:Drum"/>
<xsd:enumeration value="ms:Dozen"/>
<xsd:enumeration value="ms:Each"/>
<xsd:enumeration value="ms:Fahrenheit"/>
<xsd:enumeration value="ms:TrackFoot"/>
<xsd:enumeration value="ms:PoundsperSqFt"/>
<xsd:enumeration value="ms:FeetPerMinute"/>
<xsd:enumeration value="ms:Foot"/>
<xsd:enumeration value="ms:Gallon"/>
<xsd:enumeration value="ms:PoundsperGallon"/>
<xsd:enumeration value="ms:GramsperLiter"/>
<xsd:enumeration value="ms:Gram"/>
<xsd:enumeration value="ms:Gross"/>
<xsd:enumeration value="ms:Grain"/>
```

```
<xsd:enumeration value="ms:GrossYard"/>
<xsd:enumeration value="ms:Hectoliter"/>
<xsd:enumeration value="ms:Hank"/>
<xsd:enumeration value="ms:HundredFeet"/>
<xsd:enumeration value="ms:HundredCubicFeet"/>
<xsd:enumeration value="ms:Bale"/>
<xsd:enumeration value="ms:BaseBox"/>
<xsd:enumeration value="ms:Bucket"/>
<xsd:enumeration value="ms:Bundle"/>
<xsd:enumeration value="ms:Beam"/>
<xsd:enumeration value="ms:BoardFeet"/>
<xsd:enumeration value="ms:Bag"/>
<xsd:enumeration value="ms:Bar"/>
<xsd:enumeration value="ms:Block"/>
<xsd:enumeration value="ms:Bulk"/>
<xsd:enumeration value="ms:Bottle"/>
<xsd:enumeration value="ms:Barrel"/>
<xsd:enumeration value="ms:Bushel"/>
<xsd:enumeration value="ms:Box"/>
<xsd:enumeration value="ms:MillionBTUs"/>
<xsd:enumeration value="ms:Centipoise"/>
<xsd:enumeration value="ms:Case"/>
<xsd:enumeration value="ms:Carboy"/>
<xsd:enumeration value="ms:CubicCentimeter"/>
<xsd:enumeration value="ms:Carat"/>
<xsd:enumeration value="ms:CubicFeet"/>
<xsd:enumeration value="ms:Container"/>
<xsd:enumeration value="ms:CubicInches"/>
<xsd:enumeration value="ms:Cylinder"/>
<xsd:enumeration value="ms:Centimeter"/>
<xsd:enumeration value="ms:Can"/>
<xsd:enumeration value="ms:Crate"/>
<xsd:enumeration value="ms:Cartridge"/>
<xsd:enumeration value="ms:CubicMeter"/>
<xsd:enumeration value="ms:Cassette"/>
<xsd:enumeration value="ms:Carton"/>
<xsd:enumeration value="ms:Cup"/>
<xsd:enumeration value="ms:HundredPounds"/>
<xsd:enumeration value="ms:Coil"/>
<xsd:enumeration value="ms:CubicYard"/>
<xsd:enumeration value="ms:Days"/>
<xsd:enumeration value="ms:Degree"/>
<xsd:enumeration value="ms:Dram"/>
<xsd:enumeration value="ms:Miles"/>
<xsd:enumeration value="ms:Decimeter"/>
<xsd:enumeration value="ms:Drum"/>
<xsd:enumeration value="ms:Dozen"/>
<xsd:enumeration value="ms:Each"/>
<xsd:enumeration value="ms:Fahrenheit"/>
<xsd:enumeration value="ms:TrackFoot"/>
<xsd:enumeration value="ms:PoundsperSqFt"/>
<xsd:enumeration value="ms:FeetPerMinute"/>
<xsd:enumeration value="ms:Foot"/>
<xsd:enumeration value="ms:Gallon"/>
<xsd:enumeration value="ms:PoundsperGallon"/>
<xsd:enumeration value="ms:GramsperLiter"/>
<xsd:enumeration value="ms:Gram"/>
<xsd:enumeration value="ms:Gross"/>
<xsd:enumeration value="ms:Grain"/>
<xsd:enumeration value="ms:GrossYard"/>
<xsd:enumeration value="ms:Hectoliter"/>
<xsd:enumeration value="ms:Hank"/>
<xsd:enumeration value="ms:HundredFeet"/>
<xsd:enumeration value="ms:HundredCubicFeet"/>
<xsd:enumeration value="ms:Horsepower"/>
<xsd:enumeration value="ms:HundredTroyOunces"/>
<xsd:enumeration value="ms:Hours"/>
<xsd:enumeration value="ms:HundredWeight"/>
<xsd:enumeration value="ms:HundredWeight"/>
<xsd:enumeration value="ms:Hertz"/>
<xsd:enumeration value="ms:InchPound"/>
```

```
<xsd:enumeration value="ms:Inch"/>
<xsd:enumeration value="ms:Joint"/>
<xsd:enumeration value="ms:Jar"/>
<xsd:enumeration value="ms:Jug"/>
<xsd:enumeration value="ms:Kilowatt"/>
<xsd:enumeration value="ms:KilogramsperCubicMeter"/>
<xsd:enumeration value="ms:Keg"/>
<xsd:enumeration value="ms:Kilogram"/>
<xsd:enumeration value="ms:KilowattHour"/>
<xsd:enumeration value="ms:KilometersPerHour"/>
<xsd:enumeration value="ms:Kilopascal"/>
<xsd:enumeration value="ms:Kit"/>
<xsd:enumeration value="ms:Kelvin"/>
<xsd:enumeration value="ms:Pound"/>
<xsd:enumeration value="ms:Linear Foot"/>
<xsd:enumeration value="ms:LongTon"/>
<xsd:enumeration value="ms:Length"/>
<xsd:enumeration value="ms:Lot"/>
<xsd:enumeration value="ms:LumpSum"/>
<xsd:enumeration value="ms:Liter"/>
<xsd:enumeration value="ms:Millibar"/>
<xsd:enumeration value="ms:Milligram"/>
<xsd:enumeration value="ms:milligram_per_liter" id="mg/l"/>
<xsd:enumeration value="ms:Metric"/>
<xsd:enumeration value="ms:Minutes"/>
<xsd:enumeration value="ms:Milliliter"/>
<xsd:enumeration value="ms:Millimeter"/>
<xsd:enumeration value="ms:Months"/>
<xsd:enumeration value="ms:MetricTon"/>
<xsd:enumeration value="ms:Meter"/>
<xsd:enumeration value="ms:Barge"/>
<xsd:enumeration value="ms:Load"/>
<xsd:enumeration value="ms:ShortTon"/>
<xsd:enumeration value="ms:OvertimeHours"/>
<xsd:enumeration value="ms:Ounce-Av"/>
<xsd:enumeration value="ms:Pages-Electronic"/>
<xsd:enumeration value="ms:Percent"/>
<xsd:enumeration value="ms:Pail"/>
<xsd:enumeration value="ms:Piece"/>
<xsd:enumeration value="ms:Pad"/>
<xsd:enumeration value="ms:Pallet"/>
<xsd:enumeration value="ms:Package"/>
<xsd:enumeration value="ms:PalletUnitLoad"/>
<xsd:enumeration value="ms:Pair"/>
<xsd:enumeration value="ms:PoundsperSqInch"/>
<xsd:enumeration value="ms:Pint"/>
<xsd:enumeration value="ms:Quart"/>
<xsd:enumeration value="ms:RevolutionsPerMinute"/>
<xsd:enumeration value="ms:Rod-5.5Yards"/>
<xsd:enumeration value="ms:Reel"/>
<xsd:enumeration value="ms:Roll"/>
<xsd:enumeration value="ms:Ream"/>
<xsd:enumeration value="ms:Run"/>
<xsd:enumeration value="ms:Trimester"/>
<xsd:enumeration value="ms:SquareMeterspersecond"/>
<xsd:enumeration value="ms:SquareMile"/>
<xsd:enumeration value="ms:SquareFoot"/>
<xsd:enumeration value="ms:Sheet"/>
<xsd:enumeration value="ms:SquareInch"/>
<xsd:enumeration value="ms:Sack"/>
<xsd:enumeration value="ms:SquareMeter"/>
<xsd:enumeration value="ms:Spool"/>
<xsd:enumeration value="ms:Strip"/>
<xsd:enumeration value="ms:Set"/>
<xsd:enumeration value="ms:Skid"/>
<xsd:enumeration value="ms:SquareYard"/>
<xsd:enumeration value="ms:Tube"/>
<xsd:enumeration value="ms:Truckload"/>
<xsd:enumeration value="ms:Tote"/>
<xsd:enumeration value="ms:GrossTon"/>
<xsd:enumeration value="ms:Thousand"/>
```

```
                <xsd:enumeration value="ms:Tank"/>
                <xsd:enumeration value="ms:ThousandFeet"/>
                <xsd:enumeration value="ms:NetTon"/>
                <xsd:enumeration value="ms:TroyOunce"/>
                <xsd:enumeration value="ms:ThousandFeet"/>
                <xsd:enumeration value="ms:ThousandSquareFeet"/>
                <xsd:enumeration value="ms:Unit"/>
                <xsd:enumeration value="ms:Week"/>
                <xsd:enumeration value="ms:Yard"/>
                <xsd:enumeration value="ms:Years"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="tQuantity">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="--- no quantity ---"/>
                <xsd:enumeration value="ms:concentration"/>
                <xsd:enumeration value="ms:temperature"/>
                <xsd:enumeration value="ms:speed"/>
                <xsd:enumeration value="ms:time"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="basicDataType">
            <xsd:choice>
                <xsd:element name="primitiveDT" type="spm:tPrimitiveDataType"/>
                <xsd:element name="derivedDT" type="spm:tDerivedDataType"/>
            </xsd:choice>
        </xsd:complexType>
        <xsd:simpleType name="tPrimitiveDataType">
            <xsd:restriction base="xsd:QName">
                <xsd:enumeration value="xsd:string"/>
                <xsd:enumeration value="xsd:boolean"/>
                <xsd:enumeration value="xsd:decimal"/>
                <xsd:enumeration value="xsd:float"/>
                <xsd:enumeration value="xsd:double"/>
                <xsd:enumeration value="xsd:duration"/>
                <xsd:enumeration value="xsd:dateTime"/>
                <xsd:enumeration value="xsd:time"/>
                <xsd:enumeration value="xsd:date"/>
                <xsd:enumeration value="xsd:hexBinary"/>
                <xsd:enumeration value="xsd:base64Binary"/>
                <xsd:enumeration value="xsd:anyURI"/>
                <xsd:enumeration value="xsd:QName"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="tDerivedDataType">
            <xsd:restriction base="xsd:QName">
                <xsd:enumeration value="xsd:integer"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:schema>
```