Managing Structural Genomic Workflows using Web Services

Maria Cláudia Cavalcanti[1,2], Rafael Targino[1], Fernanda Baião[1], Shaila C. Rössle[3], Paulo M. Bisch[3], Paulo F. Pires[2], Maria Luiza M. Campos[2], Marta Mattoso[1,*]

[1] Systems Engineering and Computer Science, COPPE - {yoko, targino, baiao, marta}@cos.ufrj.br

[2] Computer Science Department, DCC/IM, NCE - {yoko, paulopires, mluiza}@nce.ufrj.br

[3] Institute of Biophysics, IBCCF - {shaila, pmbisch}@biof.ufrj.br

UFRJ- Federal University of Rio de Janeiro, Brazil

Abstract

In silico scientific experiments encompass multiple combinations of program and data resources. Each resource combination in an execution flow is called a scientific workflow. In bioinformatics environments, program composition is a frequent operation, requiring complex management. A scientist faces many challenges when building an experiment: finding the right program to use, the adequate parameters to tune, managing input/output data, building and reusing workflows. Typically, these workflows are implemented using script languages because of their simplicity, despite their specificity and difficulty of reuse. In contrast, Web service technology was specially conceived to encapsulate and combine programs and data, providing interoperation between applications from different platforms. The Web services approach is superior to scripts with regard to interoperability, scalability and flexibility issues. We have combined metadata support with Web services within a framework that supports scientific workflows. While most works are focused on metadata issues to manage and integrate heterogeneous scientific data sources, in this work we concentrate on metadata support to program management within workflows. We have used this framework with a real structural genomic workflow, showing its viability and evidencing its advantages.

Keywords: Web services, scientific workflows, workflow management, distributed architectures, metadata

* Corresponding author.

1. Introduction

In the last decades, scientific laboratories have performed *in silico* scientific experiments – along with their traditional *in vitro* – which raised the concept of an e-scientist. In this context, computer programs and electronic data became some of the most valuable scientific resources used by e-scientists during their experimentation process.

A scientific workflow can be defined as a combination of data and sequences of programs. Once executed, this sequence of programs with its inputs and outputs characterize an experiment. The main scientific resources of an experiment are: data, program, and workflow. The possibility of performing *in silico* experiments through multiple combinations of programs and data resources makes their management a complex task. This is particularly true in the bioinformatics scenario where many programs are legacy or third-party code and data resources come from public databases.

Ideally, scientists should be able to configure their own bioinformatics workflows by dynamically combining programs provided by different teams, finding alternative programs to choose from, tuning workflow programs by trying different sets of parameter values, and running partial executions of the workflow. Moreover, once the workflow is defined, the scientist should not be concerned with program changes, nor manage transformations from data output to data input along the program chains. Finally, all these workflow executions should be registered, and made available for queries and for reuse.

However, this ideal scenario is far from reality. Due to the recent growth in program and data availability, the current organization of scientific labs cannot cope with managing this amount and diversity of valuable resources. There are typically three main issues: (i) resources are disorganized and scattered, (ii) resources are not described, and (iii) resource composition is very specific.

Resources become disorganized and scattered since scientific labs usually have multiple versions of a single program, as well as multiple formats of a data set (images, flat files, databases, etc.). Typically, each program is studied and installed by one or more scientists, who are responsible for providing specific configuration for each workflow. Also, multiple data sets are used as input to those programs. Each of these data sets has a particular format and has been prepared by a scientist responsible for a specific experiment. These programs and data can be distributed on the Web and the programs can also be third-party code.

The lack of resource description is evidenced by the way scientists organize resources. Usually, they rely on the local file directory structure to organize programs, data inputs and outputs, labeling them according to the experiment or scientist. However, the lack of semantics and data structures to manage these resources makes it hard to identify related or similar programs and data. All these versions of programs and data, different formats and workflows are not easy to manage. It is hard to find out that a set of programs can be just different code versions of a same program. Even under rigid lab rules it is very difficult to keep track of these resources.

Script languages are frequently used by biological communities to implement these scientific workflows. Nevertheless, resource composition with scripts is very specific. Scripts provide an easy way to automate program calls and they are especially useful for parameter pre-setting. Usually, biological programs come with ready-to-use scripts, where parameters are pre-set with default values. When choosing to use these scripts, biologists do not need to learn program configuration details. Thus, scripts are very specific and hard to reuse. Also, the scientist might not be using the best program configuration. Another problem, according to the challenges presented by Bhowmick *et al.* in [12], is that scripts are often complex to read and very dependent on the programmers who wrote them.

In summary, workflow definition is a frequent operation in bioinformatics environments. To build an

experiment, the scientist is faced with many challenges, such as finding the right programs to use, the adequate parameters to tune, managing input/output data, building and reusing workflows. All these issues increase the complexity of workflow management.

This work proposes the use of the Web services technology associated to metadata to describe and manage scientific resources. Web services [58,59] provide a published interface to data or program resources facilitating workflow management and execution. Metadata can be defined as descriptive information about the structure and meaning of a resource. Although originally applied to data resources, it may refer to any kind of resource, such as applications or processes [43]. We have specialized the Web services technology to add metadata to the Web services program and data publication interface. To manage those resources we have defined navigation, publication and experimentation modules coupled to the Web services architecture [58].

The idea of having a Web service instead of a script for biological programs encapsulation and composition is particularly interesting with respect to interoperability. Web services technology was specially conceived to provide interoperation between applications from different platforms. Programs can be encapsulated in services. Workflows are then composed by service calls and not direct program calls. If a program location changes, only the service description is changed and all workflows that use this program are immune to this change.

Web services are an appropriate solution to support geographically distributed team collaboration through resources reuse and exchange. Particularly, Web services have been pointed out in the bioinformatics area as a potential technology to allow biological data to be fully exploited [51], and many bioinformatics resource management projects are moving towards Web services technology [64,45,61,13,32,19].

However, Web services alone do not provide enough semantics to support bioinformatics program composition. We have combined metadata support with Web services within a framework for scientific resource management, named SRMW (Scientific Resource Management based on Web services).

While most works are focused on metadata issues to manage and integrate heterogeneous scientific data sources, in this work, we concentrate on metadata support for program management within workflows. This work contributes by showing how the Web services technology can be effectively applied to build and execute scientific workflows with the help of metadata support. Thus, data integration initiatives are complementary to our work.

We illustrate our approach in a real bioinformatics scenario using the MHOLline workflow [47]. We compare Web services against the scripting scenario, showing the advantages of workflow construction with Web services, and presenting experimental results regarding their execution times, showing the feasibility of our proposal in a real application. We also present the adequacy of our metadata support to describe scientific resources and show how these descriptors are incorporated to the Web services technology.

This work is organized as follows. Next Section defines workflows and briefly presents the Web services technology used in this work. Section 3 presents MHOLline, a structural genomic workflow that is used as a target application for our proposal, while the fourth Section illustrates how it can be used under Web services technology. Section 5 shows the SRMW architecture with its Web services metadata support, while Section 6 illustrates how the SRMW can be instantiated in the MHOLline application. Section 7 discusses related work in the literature. Finally, Section 8 concludes this paper and points to future work.

2. Workflows and the Web services technology

We define a workflow as a collection of tasks organized to accomplish some scientific experiment. A task can be performed by one or more software systems. Examples of tasks include executing a program, updating a file or database, filtering a dataset or transforming it. In addition to a collection of tasks, a workflow defines

the order of task invocations or conditions under which tasks must be invoked, the task synchronization, and the information flow (dataflow) among tasks. Therefore, a workflow W is represented by a quadruple (*T*, *V*, *Sf*, *Cf*) where:

T is a set $\{t_1, t_2, \ldots, t_n\}$ of tasks of *W*,

V is a set of variables $\{v_1, v_2, \ldots, v_n\}$ of *W* defining the dataflow,

Sf is a successor function associated to every task $t \in T$, and

Cf is a condition function associated to every task $t \in T$.

The Web services technology provides the building blocks for managing workflows over a set of heterogeneous and distributed services available on the Web. Web services provide a published interface to data or program resources. They are implemented as modular programs, generally independent and self-describing, which can be discovered and invoked across the Internet or an enterprise intranet. Like components, Web services expose an interface that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web services are not accessed via protocols dependent on a specific object-model. Instead, Web services are accessed via ubiquitous Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP) [25] and XML [57], which are vendor independent.

Web Services Description Language (WSDL) [55] is an XML-based language to describe the interface of a Web service, thus enabling a program to understand how it can interact with a Web service. Each Web service publishes its interface as a WSDL document (an XML document) that completely specifies the service's interface so that client applications can automatically bind to the Web service. The SOAP [56] protocol extends XML so that computer programs can easily pass parameters to server applications and then receive and understand the returned semi-structured XML data document.

Since the Web services technology uses XML as the encoding system, data is easily exchanged between computing systems with distinct architectures and distinct data formats. While WSDL completely describes the Web service interface, SOAP completely describes parameters, data types and exceptions included in a message being exchanged between Web services.

The Web services architecture (illustrated in Figure 1) provides the necessary mechanisms to enable interoperability among heterogeneous resources in the Web. A Web resource can be accessed by any type of client program on the Web once its interface is published as a WSDL document, in a Web services registry. E-scientists can make use of such a feature to create and expose a set of common interfaces for accessing biological programs such as BLAST [43] and MODELLER [48]. Exposing those programs as Web services hides their heterogeneity and complexity facilitating the task of interacting with such resources.

Besides providing interoperability among Web resources, the Web services technology also provides the necessary mechanisms to define workflow processes through the composition of basic Web services. Such compositions of Web services are defined through XML-based languages expressing the flow of control and data across a collection of Web services whose choreography performs a workflow. Currently, there is no agreement upon a standard language for composing Web services. There are different language proposals within the Web services technology, such as WSFL [40], XLANG [52], WSCI [54], and BPML [3]. Recently IBM, Microsoft, and BEA released BPEL4WS – Business Process Execution Language for Web services [20]. BPEL4WS is an XML-based language for coordinating business processes over the Web, which relies on the Web services technology. BPEL4WS actually replaces IBM's WSFL and Microsoft's XLANG specifications. 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 building Web services based workflows. BPEL4WS provides a language to formally specify business processes and business interaction protocols. It extends the interaction model of WSDL to define a process that provides and consumes multiple Web service interfaces.

The BPEL4WS language has several constructors that are used to manipulate processes, data input and output, execution flow, and the mapping of data between the processes along the workflow steps. For

example, one process can be the initial step responsible for processing the input data of the workflow or it can represent the invocation of a Web service that is part of the workflow. Examples of workflow constructors in BPEL4WS are: <receive>, <invoke>, <wait> and <reply>.

Each process can contain parameters, data input and output. Data is represented in BPEL4WS through the constructor <container> and are used to connect the data output of a process to the data input of the next. This is done through the constructor <assign>. The execution flows are the connections between processes and define their order of execution. This flow can specify a sequential or parallel execution, or even define conditions to drive alternative execution flows. The BPEL4WS flow constructors are <sequence>, <switch>, <flow> and <while>. Finally, the external workflow processes (Web services) are defined through the constructor <partner>. More details and examples are given in Section 4.

Despite the many advantages of the Web services technology, it alone is not enough to assist the development of real scientific workflow scenarios. Besides tools for resolving interoperability and composition issues, there is also the need to address semantic interoperability among scientific resources. The scientific resource semantics define its meaning, so that a client can infer that such a resource has the capability to answer his specific needs. Web services technology must be complemented by other technologies, such as domain modeling [17] or ontologies [64,49], in order to build an environment for composing heterogeneous and distributed resources, as is the case of scientific workflows. Semantic issues are discussed in Section 5.

3. The MHOLline workflow

MHOLline [47] is an example of a bioinformatics workflow for structural genomic projects that is not easy to build. Its sophisticated set of programs makes it difficult to combine the outputs and inputs of each program. Thus, a package solution for the MHOLline workflow is under development at IBCCF/UFRJ. The same workflow is proposed as a Web services solution in Section 4.

Structural genomic projects are producing a vast amount of protein sequences as data resources, emerging the need of 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, isolated sequence homology searches do not always provide all of 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 protein three-dimensional structures is comparative modeling, which predicts the most 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 modeled; selecting related sequences as templates; aligning the sequence with the templates; building a model; and then validating the structure. There are several programs addressing each of these steps. To enable large-scale modeling we are developing a workflow that assembles these steps in an automated program sequence.

MHOLline combines a specific set of programs for the comparative modeling approach. For template structure identification, it uses the BLAST algorithm searches [43]. A refinement in the template search step was implemented by a program called BATS (Blast Automatic Targeting for Structures) [47], 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 MODELLER [48], and models are evaluated using PROCHECK [38] scores. The MHOLline workflow is illustrated in Figure 2.

4. The MHOLline workflow supported by Web services

Through our Web services framework, programs and data can be published and further browsed to build a composition of programs generating simple scientific workflows that can also be published. The programs involved in the current MHOLline workflow definition (BLAST, BATS, MODELLER) were encapsulated into Web services (future versions of our implementation will also include the PROCHECK program). An additional Web service was developed to combine all these services into MHOLline, as shown in Figure 3.

The Web services were deployed in a Web service provider, to allow applications other than the MHOLline workflow to automatically bind to the available Web services. It is worth noting that these programs were legacy programs with no source code available. However, when encapsulated into a Web service, they can become available in the Internet.

All Web services were implemented and published using Apache Tomcat 4.0.4 [2] powered by the AXIS engine [6], which processes SOAP messages. For the workflow, we used IBM BPWS4J 1.0.1 [33] to process the MHOLline specification, written in BPEL4WS.

4.1 MHOLline programs as Web services

We have defined WSDL documents for BLAST, BATS, and MODELLER programs. Figure 4 shows an example for the BLAST program. Note that the port type `BlastRunner` includes the `runBlast` operation, which refers to its input and output messages `runBlastRequest` and `runBlastResponse`.

The declarations in Figure 4 define how the SOAP messages will be generated when the BLAST Web service is invoked. The generated SOAP request message is shown in Figure 5. Besides such WSDL definitions we have also developed glue code for mapping input and output data between Web services and legacy programs.

4.2 Defining the MHOLline workflow with BPEL4WS

The BPEL4WS language was used to formally specify the composition of the programs defined in the MHOLline workflow specification, where each program (service) is defined as a <`partner`> (Figure 6).

<`container`> tags were used to reference the messages defined in the Web services, as in Figure 7.

A program composition in the MHOLline workflow is defined using a <`sequence`> tag, which defines how the partners will be sequentially executed. The specification of a sequence includes definitions of the input message (the <`receive`> tag), of the service invocation (the <`invoke`> tag) and of the output message (the <`reply`> tag), as exemplified in Figure 8.

The <`assign`> tag establishes the relationship between the output message of a service and the input message of the service in the workflow, as illustrated in Figure 9.

Similarly to the individual programs, the MHOLline Web service is also described in a WSDL file (Figure 10). Messages, port types and operations are also defined, and there is an additional element <`serviceLinkType`> to define the role of each service during their interaction.

4.3 Comparing Web services with scripting approaches

This Section compares the Web services approach against the traditional scripting scenario for the MHOLline workflow. We show the advantages of the workflow construction supported by Web services, and present experimental results of both workflow executions.

In the scripting scenario, the MHOLline workflow was defined through a Perl script, in which each external program was invoked locally, through the file system path. In the Web services scenario, the workflow Web service and the Web services that wrap the external programs were installed in the same machine and were invoked by a local Java application.

4.3.1 Comparing MHOLline workflow definitions

In the script language approach for the workflow definition, MHOLline was implemented as a Perl script with system calls to the executable files of each composing program (BATS, BLAST and MODELLER). Since these system calls work on top of the operational system file manager, they should indicate the complete program location paths (Figure 11-B). Workflow input data was set by indicating the path of the FASTA file that was passed as input to the first workflow program, BLAST (Figure 11-A). Output files generated by each program were saved to the local disk, and used as input to the next program in the execution flow. Some output files needed parsing to extract only the relevant data embedded in them (Figure 11-C). Also, conditional execution flows in the workflow definition were implemented using the available commands of the script programming language (Figure 11-D).

The script language approach is adequate in cases where the script is constructed and executed by only one person, working in the same machine throughout the process. However, in real world scenarios, it frequently happens that multiple users need to share workflow definitions and perform several executions of the same workflow. Moreover, each user may have his/her set of input data and parameter values to work with. These situations are very common in real bioinformatics laboratories, and they turn script construction, maintenance and sharing into very hard and time-consuming tasks.

More specifically, script languages were not designed as program composition languages, and therefore important information is hidden in their structure. For example, the set of programs, their input and output data, as well as their execution flow are implicit in the script, and it is very hard for someone who is not familiar with the script language to understand it clearly. Directory locations, file names and even parameter values may be hard-coded, which increases the difficulty in performing adaptations or modifications in the workflow definition, whenever needed. Moreover, even though Perl is an interpreted language, Perl scripts are not fully portable among different operational systems.

In the Web services approach, the workflow is defined using BPEL4WS, which is a specific-purpose high-level program composition language. Each composing program, wrapped into a Web service, is invoked in a unique way, through the <invoke> constructor (Figure 11-F). Data transfer between two subsequent Web services in a workflow definition is explicitly declared using specific data structures and constructors (Figure 11-E). Boolean conditions and conditional execution flows are also explicitly defined (Figure 11-G).

The Web services approach facilitates MHOLline workflow construction and maintenance processes, since they rely on a higher-level specification language that explicitly represents programs execution flow. Therefore, it makes it easier to address tasks such as adding/removing programs to/from the workflow definition, defining compensation activities for error handling routines, and detecting possible system faults.

The proposed approach also enables the workflow builder to provide semantic information, such as program grouping into categories, to the workflow definition (for example, different versions of the BLAST program: NCBI, Wu, local copy). The system could use this information to automatically choose, among programs of the same category, the most adequate program to perform a workflow step, according to some criteria (performance, distance, monetary cost, etc.) [4]. Additional mechanisms to enable partial workflow re-executions, authentication, data/program access security and logging could easily be coupled to improve this approach.

Graphical tools are available and may be used to help in the workflow definition and the BPEL4WS file generation, such as the IPB plug-in for the Eclipse interface [21], and the VisualScript XML commercial tool [53].

Summing up, exposing MHOLline programs as Web services facilitates access to such programs, improving the communication *interoperability* among them. BPEL4WS provides a *flexible* way of specifying Web services compositions, while it improves scientists' *productivity* by simplifying workflow definitions.

4.3.2 Comparing MHOLline workflow executions through experimental results

Every step of the MHOLline workflow was automatically executed, with no human interaction. BPEL4WS is an expressive language, providing additional commands to verify data input values, to start parallel executions and to allow workflow shortcuts. Through a Web interface the user may tune parameters and interact with the workflow definition.

Figure 12 illustrates the MHOLline workflow processing for a molecular sequence. It starts by sending a message to the workflow Web service, including the molecular sequence. The message is received and forwarded to the first workflow task (the BLAST Web service), which is then invoked. After processing the sequence, BLAST program results are returned to the BLAST Web service, which forwards them to the BATS Web service through the BPEL4WS workflow engine. Similarly, BATS is invoked, and its results are forwarded to the MODELLER Web service. Finally, MODELLER results are then returned to the workflow Web service, and then to the caller.

In order to evaluate the performance of our approach, we have run an experiment comparing the results and the execution times of the MHOLline workflow in two scenarios: in the traditional scenario, in which the MHOLline workflow is defined through scripts; and the proposed scenario, in which the MHOLline workflow is defined through Web services technology.

The evaluation does not consider the execution time of the involved programs (BLAST, BATS e MODELLER) since they are the same in both scenarios and we are interested in comparing the overhead introduced by the utilization of the Web service technology. The MHOLline workflow was executed ten times in each scenario, having as its input a FASTA file (shown in Figure 13) containing only one protein sequence with 120 amino acids. The final workflow result (the molecular structure) was exactly the same in both scenarios, as expected.

The experiment was carried out in four steps in a PC workstation with one AMD Athlon XP 1800 processor and 512Mb of RAM memory, running Windows 2000. In the first step, the time between the beginning of the workflow execution and the dispatch of the BLAST invocation was measured. In the second step, the time between receiving the BLAST response and the dispatch of the BATS invocation was measured. The third step was essentially the same as the second step but it considered receiving the BATS result and the invocation of the MODELLER program. The last step has measured the time between receiving the MODELLER result and the end of the workflow execution.

As expected, the execution of the scripting scenario was faster than the Web services scenario, as shown in Table 1. However, although the execution times of the Web services approach are on average one order of magnitude greater than the scripting approach, this result shows the feasibility of the Web services approach. This conclusion arises from the fact that the total execution time of the MHOLline workflow, i.e. the execution time including the execution of all its external programs, is considerably greater than the overhead introduced by the Web services. For instance, the MODELLER program takes in average 245 seconds to run (Table 2). Since the average overhead of the Web services approach is 2.2 seconds, it is less than 1% of the time spent to run the MODELLER program. Moreover, since the experiment was carried out in a single machine, network costs were not considered. In a distributed scenario, the programs would run in remote machines, and therefore the network latency would increase the total workflow execution time in both scenarios. Thus, the overhead of the Web services approach would be even more insignificant.

Therefore, one can consider that it is totally feasible to implement the Web services approach in a production environment and that the apparent overhead of such solution would be entirely compensated by the benefits previously mentioned in Section 4.3.1: flexibility, interoperability, and productivity. MHOLline is not the only workflow that might be available for scientific users. Different configurations of MHOLline workflow, using different parameter values or using alternative programs, would also be useful. Fortunately, with the Web services approach proposed in this work, such changes could be easily

accommodated, which would be harder in the script language approach.

It is worth noting that the proposed solution is flexible enough for building other scientific workflow definitions. Moreover, once defined, they can be published as services and the MHOLline definition as a whole can be incorporated into other workflow definitions, without any code modification.

5. Metadata support in the SRMW architecture

Although Web services technology is on the right track to support a full-featured e-scientist laboratory, its description mechanisms are not enough to provide semantics to the resources being published. The overall computational environment should include mechanisms for description and management of scientific resources through an architecture that couples metadata with these resources. In this work we briefly present the SRMW (Scientific Resource Management based on Web services) architecture (Sub-section 5.1) and its metamodel (Sub-section 5.2), the SPMW (Scientific Publication Metamodel for Workflows), designed for the publication of scientific description information. We also discuss how SPMW is coupled to WSDL, in order to provide more semantic information to WSDL documents (Sub-section 5.3). More details on this framework can be found in [15]. The SRMW framework is an evolution of our previous scientific metamodel [17] and architecture [16,18] to the Web services approach.

5.1 Scientific resource management based on Web services

The main scientific resources supported by SPMW are: data, program, model, workflow, experiment and essay. Programs and data resources can be viewed according to different levels of abstraction. In the scientific environment, these different levels of abstraction correspond to different objects that are manipulated at different moments. Programs and data can be viewed at an operational level, when the user deals with code executions, and at a descriptive level, when the user deals with the information that represents them. A program can be viewed at the descriptive level as a textual specification or as an interface description. At the operational level, a program can be viewed as the execution code generated by the compilation of a program specification. Analogously, data sets (files) are at the operational level, while the information about them is at the descriptive level. Models, workflows, experiments and essays are all descriptive resources, and have no corresponding object at the operational level.

In the SRMW architecture there is a clear distinction between these two levels: operational and descriptive. Therefore, we have added two new resources to our list of scientific resources: code and data category resources. From now on, the program resource refers to the program interface, while the code resource refers to the program execution code. Also, the data resource refers to data sets, while the data category resource refers to the data set descriptions.

The SRMW architecture (Figure 14) has five modules. There are two main modules to manage scientific resources: the Web services provider module and the Web services registry module. The provider module deals with operational resources, i.e., data and codes. The registry module plays the role of a metadata repository manager, dealing with descriptive resources, i.e., data category, program, model, experiment, and workflow descriptions. The three other modules interact with those two main modules and interface with the user: Publication, Experimentation and Navigation. The Publication module is used for describing scientific resources. The Navigation module is used to browse scientific resource corresponding descriptions. Finally, the Experimentation module is used to execute and track *in silico* experiments.

In the SRMW architecture, there are two main roles: the publisher and the user. The user navigates through descriptive resources, trying to find some useful resource, and then actually accesses it, performing experiments. On the other hand, the publisher is a resource provider. Frequently, one person plays these two roles.

The SRMW architecture dynamics is based on the interaction with users and publishers. First, the publisher provides the operational resources (code or data) he wants to publish through Web service provider modules. This means to build a Web service – and its corresponding WSDL document – for each

code and data resources, so that they can become available to Web users. Then, the publisher interacts with the Publication module (step 1) to describe those operational resources, generating specialized WSDL documents. These documents include more semantic information about code and data, according to the SPMW metamodel, such as the mathematical model implemented by a program and data descriptions at a higher level. After validated, these documents are then stored by the Web service registry module.

Then, the user interacts with the Navigation module (step 2) to find the resources he needs. As he finds what it seems to be appropriate, he assumes the publisher role and interacts with the Publication module, which helps him on planning an organized execution (step 3) through a workflow definition. The user then interacts with the Experimentation module (steps 4, 5 and 6) where he/she is able to choose, instantiate and execute workflow specifications, which refer to program resources. When the workflow is completely instantiated, the Experimentation module issues requests to the Web services providers (steps 7 and 8), which actually access data sets and execute code. It works as a Web services requester that builds user requests to get input data and to run a specific code on such data. Data results are also available as a Web service (a user choice of temporarily or definitely). Moreover, the Experimentation module keeps track of the ongoing experiments, by registering each essay (workflow instance). For instance, it is possible to keep track of which data was used as input to a specific code execution, and also, which code has generated some specific data.

5.2 SPMW metamodel

Differently from approaches based strictly on ontologies, in a metamodel-based approach the concepts are captured and explicitly represented as the basis for structuring resource descriptors. The SPMW metamodel allows for explicit semantic representation of scientific resources. It includes descriptions for scientific model, program and code, among others. The remainder of this Section presents the set of concepts that forms the SPMW metamodel. Figure 15 presents the UML representation of a simplified version of SPMW, where data, code, program, model, workflow and experiment concepts can be visualized.

In SPMW, data and code resources are identified as operational resources, which have a Web address. An operational resource is specialized into a data resource (DataResource) or a code resource (CodeResource), which are described (describedBy) by a certain “type”. In particular, 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 is an executable program instance at a specific location.

Usually, a scientific program is the implementation of a theoretic model. We believe that it is important to describe the program and the theory behind it, i.e. 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 semantics to facilitate the decision regarding its adequacy to the problem in hand. Both model and program concepts have many characteristics in common, even though they belong to different usage (and semantic) levels. The program is actually executable, while the model is a generalization of a program. Their similarity resides mainly on their relationships to other concepts, since both are associated to a set of I/O data, parameters, and constraints. To represent this similarity, both model and program can be viewed as transformations.

A transformation is a description of a data transformation process that requires input data and produces output data. Therefore, a transformation should be associated to at least one input and one output, and each I/O data refers to a data category. 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. The user may publish a code by associating it to a specific transformation, i.e., program interface. This will help the user to access, understand and use such code.

A DataCategory describes scientific data that have some common characteristics. A set of attributes is used to describe each property of a DataCategory. A DataCategory can be associated to a model (ModelDC) or to a program (ProgramDC).

In addition, we can say that a Program implements a Model. Therefore, a mapping function should exist

between a Program and a Model, meaning that for each I/O Data associated to the Program, there may be a related I/O data associated to the Model. Analogously, we may say that a ProgramDC implements a ModelDC, by establishing a mapping function between both DataCategories.

A Parameter is a concept that can represent: a model parameter used to "tune" the model for a specific objective; a processing parameter used to determine some performance and accuracy aspects; or a condition parameter used to express a value required by some program constraint.

The scientific process is strongly based on experimental investigation, evidence accumulation and result assimilation. Therefore, the use of scientific resources should also be captured by SPMW. Other advanced concepts must be identified to provide for the resource usage registration, such as the Transformation concept. A Transformation is used to describe a program. However, to make it available for an experiment, 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 text document with a workflow description language like BPEL4WS. Such workflow specification describes how the related transformations are to be processed, i.e., which transformation should be performed first, which ones can be performed in parallel, which transformation output data corresponds to another transformation input data, etc.

An Experiment is associated to a set of controlled workflows, which are usually similar to each other, and their results may be compared to each other, to verify or not the experiment hypothesis. In the case of our *in silico* laboratory environment, 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 can be mapped to an instantiation a specific workflow, as this workflow is composed by 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 (Parm match) and for each program I/O data, which data resource was used (Data Match). 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. In summary, the code execution automatically documents the use and generation of data resources as data I/O of a code resource, during an *in silico* essay.

5.3 Adding SPMW metamodel concepts to a WSDL document

To take advantage of the Web services technology we have implemented our metamodel specializing WSDL. Considering that services and programs are equivalent concepts, both the SPMW metamodel and the WSDL schema share some representation intersection. 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 not to modify WSDL metamodel but rather to add the SPMW semantic elements through specialization. Thus, we have expressed SPMW as an XML Schema.

Analogously to WSDL, we have created a root element that is composed by SPMW definitions, called `ScientificResourceDefinitions`. Each scientific resource, i.e., program, model, program/data categories, etc., must be declared under this element. The relationships between SPMW elements and WSDL elements are represented in the SPMW XML Schema as extra attributes whose values should point to WSDL element instances.

To illustrate the SPMW XML schema, in this Section we detail one of its main elements, the program element (`spm:Program`). To facilitate the visualization, instead of the usual XML textual format, we have used the hierarchic diagram shown in Figure 16 to represent the program element. In this diagram, 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. The inherited elements appear first as a separate group. Elements with a complex structure are indicated by the plus (+) sign. Each of the SPMW main elements, including the program element, contains an extensibility element (`any` element), allowing new sub-elements to be included in instance documents.

The `program` element inherits some attributes from the abstract type `tTransformation`, which are included in the first group of elements in Figure 16. `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 the constraint in natural language, and an `expression`, used to describe it in a formal language.

In the second group of elements, there are `program` specific descriptive elements. The `implementationLanguage` element holds information about 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, there is the `implements` element, which is used to refer to the `model` element, informing the relationship between a program and the theory behind it.

The specialized WSDL document includes SPMW definitions, using the WSDL `definitions` element. To illustrate how we added SPMW elements to WSDL let us take the SPMW `program` element. In a WSDL document, the `portType/operation` element corresponds to the SPMW `program` element. In a specialized WSDL document, the `program` element refers to the `portType/operation` element through its `wsdlElementRef` sub-element.

The WSDL file is duplicated by the SRMW architecture, and altered to include 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 by SRMW as a specialized WSDL document. For instance, when queried about a specific program, SRMW would provide a WSDL document with specialized semantics, including contents of all related SPMW documents.

6. The MHOLline workflow supported by SRMW

This Section shows how the SRMW architecture was used with the MHOLline workflow. Since one of the goals of the MHOLline project is to enable large-scale modeling by assembling programs on an automated workflow, the use of the SRMW architecture enables biologists to obtain structural prediction without depending upon individual bioinformatics specialists.

Figure 17 illustrates the SRMW functionality, through the Navigation, Experimentation and Publication of MHOLline resources. First, we have published useful resources to build the MHOLline application. The BLAST algorithm was published as an algorithmic model (Figure 17-a). Some other resources were also published: all the programs involved in the MHOLline application, a code resource for each published program, the MHOLline workflow itself in which those programs take part, and some available data resources.

Figure 18 shows an SRMW screen shot with the publication of Blast-P program and its association with the BLAST algorithm previously published (`implements`). The description of this program shows that it was written in C by NCBI programmers, and that its current version is 2.2.4. Also, its publication involves the description of its data input/output, through its association with input and output data categories. For instance, it receives some `protein query sequences` as input, which is described by the `protein target sequences` data category. Finally, the program is also related to a WSDL document through the reference to an instance of a `portType` operation named `blastpOperation`.

The MHOL3D is an ongoing experiment that is being conducted at IBCCF, where the MHOLline workflow is already in use. In the context of SRMW, this experiment is supported by the Navigation, Publication and Experimentation facilities. We suppose that the specialist is looking for a workflow resource to perform some essays. After navigating through all the resources related to a protein sequence (Figure 17-b), the specialist finds out that the MHOLline workflow is what he needs to use, and could then start describing the MHOL3D experiment by associating it to the MHOLline workflow.

Then, having the experiment published, the scientist may start "running" essays by providing all program input data and parameter values. Figure 17-c shows the instantiation of the MHOLline workflow at the Blast-P step, where an input screen asks for the parameter values. In addition, the Experimentation facility helps the user on choosing the adequate code, considering those code resources associated with the Blast-P program, and on choosing the input data, considering those data resources that are compatible to the Blast-P program input data category. Finally, the scientist is now able to "execute" the experiment.

During the experiment, the Experimentation module automatically updates the MHOL3D experiment document, registering all the related essays. Once the experiment results confirm the experiment hypothesis (i.e., this current MHOLline experiment execution provides good quality 3D-structures), the specialist may then decide to finish the MHOL3D experiment, completing the corresponding report. The MHOL3D experiment will then be available for other scientists, who will learn about it through the Navigation module.

In the context of the MHOL3D experiment, the need for defining *ad hoc* workflows is quite frequent. In this case, the SRMW architecture can help on the composition of these workflows. For instance, suppose that a generic structural genomic workflow was defined in terms of scientific models (a model-based workflow). The specification of this workflow would be defined as a sequence of interconnected models (Figure 2): the BLAST algorithm, the BLAST filter, the 3D structure generator, and the structure evaluator. Then, the MHOLline workflow would be a program-based workflow that implements this model-based workflow. The program-based one could be: the Blast-P program, the BATS program, then MODELLER and PROCHECK programs. If this specific set of programs defined in MHOLline is not adequate for a specific request, a new workflow should be built. Suppose, for instance, that the BATS program should be replaced by another program. 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 that an alternative to BATS would be to use the MSPCrunch program. Then, browsing the available programs (Figure 17-b), and their corresponding data categories, the scientist can build an adequate program-based workflow for the new experiment.

The current SRMW Web interface is limited to serial flow executions. We are working on coupling workflow editors with SRMW so that all features of BPEL4WS can be automatically generated. We are also working on establishing code association as the workflow is being executed.

In SRMW, users have to include a large amount of scientific information before starting to benefit from them. However, once the SRMW architecture is fully instantiated in a specific real scenario, the scientific activity is facilitated, and users have just to fill-up parameters and provide data resources. Consequently, scientific resources are automatically documented, integrated, inter-related and made consistent.

The use of the SRMW infrastructure with the MHOLline workflow confirmed the importance of some modeling concepts such as models and programs, as well as semantic levels of resources including essays and experiments.

7. Related work

In scientific laboratories, the use of the right program and data resources requires browsing previous experiments. More specifically, to understand how to use a program and set its parameters is a hard task. Therefore, it becomes difficult to choose the adequate parameter values to use, or to identify alternative programs. Accessing previous experiments is particularly useful because it allows users to retrieve

examples of program parameter settings and to consider similar programs as alternatives to solve the problem in hand. Therefore, metadata is needed to describe these programs and data.

The main contribution of our work is to combine metadata support to manage scientific workflows with the Web service technology. Thus, in this Section we discuss related work with respect to database mediation integration, grid computing, workflow management, and Web services technology in the bioinformatics scenario.

Many projects have been working on metadata to capture data lineage [7,12,24,26,27,29]. Systems such as K2/Kleisli [21] and TAMBIS [45] use the mediation approach to integrate several databases. Mediators have been traditionally used to address business database integration problems [61]. A mediator is responsible for translating query commands and query results to and from each database component being integrated, thus coming up with a unique logical view of the database. The data lineage and mediation approaches, however, do not deal with data transformations along a program chain, which is the case of a workflow.

Many scientific workflow systems are adopting grid platforms [15,26,22,36]. Grids are beginning to be used by the bioinformatics community [7,8,9]. The Open Grid Service Architecture (OGSA) [37] specification combines Web services to the grid computing platform through the Grid Service. However, those projects are currently using the grid without Web services. In addition, their focus is on resource allocation for workflow execution planning, rather than semantic issues for workflow design. The myGrid project [29] is an exception in this scenario and thus is further detailed at the end of this Section.

Workflow Management Systems (WfMS) are automated coordination engines that control workflow specification, instantiation, execution, auditing and evolution. Most of these WfMS were built to address traditional business workflows, but some projects like ZOO [34], WASA [41] and IntelliGEN [35] are using tools/environments to address scientific workflows. IntelliGEN is a workflow implementation created to schedule and support activities in a genomic laboratory to discover protein-protein interactions. IntelliGEN is built on top of METEOR Workflow Management System [49]. The architecture of METEOR includes services to build, store, execute and manage workflows. Workflow specifications are stored as XML-documents, assuming no particular implementation of the workflow runtime service. However, tasks are deployed and executed in proprietary UNIX servers, and the workflow specifications are static, that is, the user is not able to add new tasks to an existing workflow definition. In other works, such as [1], the authors give special attention to input and output data representation, allowing the design of richer workflow specifications. However, although their metamodel captures program executions, it does not consider that a set of workflow instances may take part in one scientific experiment. Also, it does not provide for model-based workflows, as it does not distinguish between model and program concepts. The BioFlow initiative [30] proposes an architecture to integrate data and bioinformatics programs. BioFlow presents facilities for the end user by proposing a graphical user interface, a workflow definition language and workflow execution engine. However, BioFlow is based on proprietary solutions. In [39], the authors propose the gRNA (Genomics Research Network Architecture) that comprises a development environment and a deployment framework to address the development of new genomic-centric applications. The gRNA deployment platform includes a Workflow API, which is a generic and configurable engine to compose a set of tasks and data into a unified process in an interpreted execution environment [13]. It provides a drag-and-drop graphical interface to enable the visual construction of workflows by the end-user. It also presents a specific module to deal with XML queries, and provides visual XML-based query interface [11]. The main contribution of gRNA lies in providing innumerous data management tools for storing, modeling, querying, and integrating heterogeneous sources of data and programs [12], using XML as an interchange format. However, none of these WfMS [41,34,35,49,1,30,39,13,11,12] use Web service technology to provide interoperability issues.

The Web services technology is an open standard already adopted by the industry. Therefore, many bioinformatics resources management projects are moving towards Web services technology

[45,51,61,64,13,32]. In [45], the authors use Web services to group and find similar biological programs, without considering their composition. The BioMoby project [61] uses Web services to publish and discover both data and programs and is thus close to our work. A special feature of BioMoby is that it is open source and provides an easy to use environment for bioinformatics developments. However, workflow composition and execution has not been described in the BioMoby environment.

The myGrid project [64] is also closely related to our work, since both explore the combination of metadata and Web services workflows. In addition, myGrid also addresses the grid platform. The composition of workflows using Web services is also part of myGrid features [63]. myGrid project [28], [29] emphasizes the symbiotic relationship between semantic Web and grid applications evidencing its leading ideas on this scenario. myGrid provides metadata support mainly through a suite of specific-domain inter-related ontologies and annotation components used to describe scientific resources. We use a metamodel to structure resources descriptions, explicitly representing metadata for models, programs, experiments and data. This is particularly suitable when describing data transformations as the metamodel clearly expresses the relationships among the resources involved. We believe that both approaches, metamodels and ontologies, are complementary [49], thus we are working to add ontology services to our metadata functionalities.

The present work innovates by proposing and showing, in a real bioinformatics scenario, the use of the Web services technology associated to metadata in a framework to describe and manage scientific resources within *in silico* scientific experiments.

8. Conclusions

The bioinformatics area is growing rapidly, thus raising many management issues to e-scientists. The vast amount of program and data resources available must be organized in an interoperable, flexible and scalable environment. In this work, we show the use of Web services as an effective infrastructure to provide such environment, i.e. an e-scientist laboratory. We discuss the role of metadata with Web services and show how it can be coupled with mechanisms for description and management of scientific resources in the SRMW architecture. We have focused on workflow issues and applied Web services in the context of a bioinformatics workflow called MHOLline.

We believe the use of Web services is on the right track to a full-featured e-scientist laboratory. If compared to the script language approach, 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. MHOLline workflow, for instance, includes Web services that interface with legacy programs written in different languages (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. Reusability is facilitated by Web services because of its modular approach. 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 they are not tied to any proprietary solution. Our experiments also show that, even though scripts are faster, the Web services overhead is negligible when considering the execution time of the workflow programs and remote program calls.

Another advantage of Web services technology is its coupling to the grid services initiative (OGSA). We are currently working on performance improvements of structural genomics workflows using parallel processing on PC clusters [42]. We intend to adapt SRMW to the grid services platform and use these workflow parallel strategies on the grid scenario.

The implementation of the Web services approach presented in this work proved to be an appropriate solution to create an environment that supports geographically distributed resource management. However, to provide a full-featured e-scientist laboratory, the Web services architecture is not enough. First, since the Web services description language was originally proposed for generic service description, it lacks application-related semantic descriptors. We have provided descriptors that can provide more

semantics to scientific applications. For instance, in the case of bioinformatics applications, we have descriptors that can provide information on the molecular sequence format (FASTA) used as input to the Blast-P program. In our approach, we have provided such descriptors by adding the SPMW metamodel constructs to WSDL. This way, metadata become explicit, and may be manipulated by user queries and navigation. One important contribution of the SRMW metamodel and architecture is to allow code and data resources to be found through their higher-level descriptions (data and transformation categories).

Also in this direction, some works try to establish domain standards for the genetic area, through the proposal of domain ontologies [64,5]. Spyns *et al.* [49] identify advantages in both ontology and metamodel approaches and propose to combine them. We are currently working on adding ontology references to our metamodel. Ontologies are important as they provide a common vocabulary and conceptualization for descriptors instances.

In addition, as in traditional scientific laboratories, e-scientists need to document their experiments. Therefore, the e-scientist laboratory should provide mechanisms for capturing and organizing experiments. In SRMW, Web services messages may be captured to provide an automatic way of registering essays. Moreover, the identification of scientific experiments in SPMW and their organization as a set of essays, allow the user to browse documented model usage.

In spite of the many advantages of the Web services technology, we believe it is important to consider it as a new and evolving technology. One drawback of the current Web services technology is the support for security. For the moment, most companies are keeping their Web services projects behind company firewalls because of lingering network security and reliability concerns. Another relevant aspect to be considered regarding the Web services technology is the real degree of interoperability among different implementations of Web services platforms. The constant evolution of Web services standards (SOAP, WSDL and UDDI), as well as the differences among multi-vendor Web services platform implementations, lead to differences between implementations of different SOAP specifications. Therefore, currently, not all Web services platforms can really seamless communicate to each other. However, this issue is already being addressed by the Web Services Interoperability Organization (WS-I) [60], which is an open industry effort chartered to promote Web services interoperability across platforms, applications, and programming languages. Therefore, we expect a higher degree of interoperability among different Web services platforms in the near future.

9. Acknowledgement

This work was partially supported by CNPq and CAPES Brazilian funding agencies. Paulo Pires is a CAPES – Brazil grant holder. The authors would like to thank the anonymous reviewers whose comments helped to improve this paper.

10. References

[1] A. Ailamaki, Y. Ioannidis, M. Livny, Scientific Workflow Management by Database Management, in: Proc. 10th International Conference on Scientific and Statistical Database Management, (Capri, 1998) 190-199.
[2] Apache Tomcat, available in http://jakarta.apache.org/.
[3] A. Arkin, Business Process Modeling Language (BPML), (Mar 2001), available in http://www.bpmi.org/bpml-spec.esp.
[4] V. Azevedo, M. Mattoso, P. Pires, Strategies for Dynamic Executions of Semantically Equivalent Web Services, Technical Report ES-637/04, COPPE/UFRJ (Brazil, 2004).
[5] M. Ashburner, S. Lewis, On ontologies for biologists: the Gene Ontology - uncoupling the web, in: Proc. Silico Biology 247, Novartis Found Symposium (2002) 66-83.
[6] Axis Project, available in http://ws.apache.org/axis/.
[7] W. Bausch, C. Pautasso, R. Schaeppi, G. Alonso, Programming for Dependability in a Service-Based Grid, in: Proc. 3rd International Symposium on Cluster Computing and the Grid (Tokyo, Japan, 2003) 164-171.
[8] Bio-GRID project, available in http://biogrid.icm.edu.pl/.
[9] BioGrid project, available in http://chipmunk.bio.indiana.edu/~gilbertd/about/nih-biogrid-mar02.pdf.
[10] R. Bose, A Conceptual Framework for Composing and Managing Scientific Data Lineage, in: Proc Int. Conf. on Scientific and Statistical Database Management (Edinburgh, 2002) 15-19.

[11] S. Bhowmick, P. Cruz, A. Laud, XomatiQ: Living with Genomes, Proteomes, Relations and a Little Bit of XML, in: Proc. 19th Int. Conf. on Data Engineering (ICDE 2003), (IEEE Computer Society, Bangalore, 2003) 857-868.
[12] S. Bhowmick, D. Singh, A. Laud, Data Management in Metaboloinformatics: Issues and Challenges, in: Proc. 14th Int. Conf. on Database and Expert Systems Applications (DEXA 2003), LNCS 2736 (Springer Verlag, Prague 2003) 392-402.
[13] S. Bhowmick, V. Vedagiri, A. Laud, HyperThesis: the gRNA spell on the curse of bioinformatics applications integration, in Proc. 2003 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2003), (New Orleans, 2003) 402-409.
[14] caBIO – The cancer Bioinformatics Infrastructure Objects, available in http://ncicb.nci.nih.gov/core/caBIO.
[15] J. Cao, S. Jarvis, S. Saini, G. Nudd, GridFlow: Workflow Management for Grid Computing, in: Proc. 3rd Int. Symposium on Cluster Computing and the Grid (Tokyo, 2003) 198-205.
[16] M. Cavalcanti, Scientific Resources Management: Towards an In Silico Laboratory, Ph.D. Thesis, Technical Report ES-605/03, COPPE/UFRJ (Brazil, 2003).
[17] M. Cavalcanti, M. Mattoso, M. Campos, F. Llirbat, E. Simon, Sharing Scientific Models in Environment Applications, in: Proc. ACM Symposium on Applied Computing (Madrid, 2002) 453-457.
[18] M. Cavalcanti, M. Mattoso, M. L. Campos, E. Simon, F. Llirbat, An Architecture for Managing Distributed Scientific Resources, in: Proc. Int. Conf. on Scientific and Statistical Database Management (Edinburgh, 2002) 47-55.
[19] T. Clark, Identity and interoperability in Bioinformatics, Briefings in Informatics 4 (1) (2003) 4-6.
[20] F. Curbera, Y. Goland *et al.*, Business Process Execution Language for Web services, V.1.0, available in http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
[21] S. Davidson, J. Crabtree, B. Brunk, J. Schug, V. Tannen, C. Overton, C. Stoeckert, K2/Kleisli and GUS: Experiments in integrated access to genomic data sources, IBM Systems Journal 40 (2001) 512-531.
[22] E. Deelman *et al.*, Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing 1 (1) (2003) 25-39.
[23] Eclipse Java IDE, available in http://www.eclipse.org/.
[24] ESP2Net Project, available in http://dml.cs.ucla.edu/projects/dml_esip.
[25] R. Fielding *et al.*, RFC 2616. Hypertext Transfer Protocol - HTTP/1.1, available in ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt (Jun 1999).
[26] I. Foster, J. Vöckler, M. Wilde, Y. Zhao, Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, in: Proc. Int. Conf. on Scientific and Statistical Database Management (Edinburgh, 2002) 37-46.
[27] J. Frew, R. Bose, Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products, in: Proc. Int. Conf .Scientific Statistical DB Management (Fairfax, USA, 2001) 180-189.
[28] C. Goble, D. De Roure, Semantic Grid: An Application of the Semantic Web, SIGMOD Record 31 (4) (2002) 65-70.
[29] C. Goble, C. Wroe, P. Lord, J. Zhao, R. Stevens, Using Semantic Concepts in the myGrid project, in: Proc. of the Global Grid Forum, Semantic Grid Workshop (2003) available in http://www.semanticgrid.org/GGF/ggf9sgws1.doc.
[30] Z. Guan, H. Jamil, Streamlining Biological Data Analysis Using BioFlow, in: Proc. 3rd IEEE Symposium on BioInformatics and BioEngineering (BIBE'03), (Maryland, 2003) 258-262.
[31] B. Howe, D. Maier, Modeling Data Product Generation, Position paper in Workshop on Data Derivation and Provenance, (Chicago, 2002), available in http://people.cs.uchicago.edu/~yongzh/papers.
[32] I3C: Interoperable Informatics Infrastructure Consortium, available in http://www.i3c.org
[33] IBM BPWS4J, available in http://www.alphaworks.ibm.com/tech/bpws4j.
[34] Y. Ioannidis, M. Livny, S. Gupta, N. Ponnekanti, ZOO: A Desktop Experiment Management Environment, in: Proc. 22nd VLDB Conference (Bombay, 1996) 274-285.
[35] K. Kochut, J. Arnold, A. Sheth et al., IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions, Distributed and Parallel Databases 13 (1) (2003) 43-72.
[36] S. Krishnan, P. Wagstrom, G. Laszewski, GSFL: A Workflow Framework for Grid Services, available in http://www-unix.globus.org/cog/projects/workflow/gsfl-paper.pdf.
[37] OGSA-DAI, available in http://www.ogsadai.org.
[38] R. Laskowski, M. MacArthur, S. Moss, M. Thornton, PROCHECK: a program to check the stereochemical quality of protein structures, Journal Appl. Cryst. 26 (1993) 283-291.
[39] A. Laud, S. Bhowmick, P. Cruz, D. Singh, G. Rajesh, The gRNA: A Highly Programmable Infrastructure for Prototyping, Developing and Deploying Genomics-Centric Applications, in: Proc. VLDB (2002) 928-939.
[40] F. Leymann, Web services Flow Language (WSFL 1.0), available in http://www-4.ibm.com/software /solutions/webservices/pdf/WSFL.pdf (May 2001).
[41] C. Medeiros, G. Vossen, M. Weske, WASA: A Workflow-Based Architecture to Support Scientific Database Applications, in: Proc. Int. Workshop and Conference on Database and Expert Systems Applications (DEXA), (London, 1995) 574-583.
[42] L. Meyer, S. Rössle , P. Bisch, M. Mattoso, Parallelism in Bioinformatics Workflows, in: Proc. Int. Conf. VECPAR, (Valencia, 2004), to appear.
[43] MDC (Metadata Coalition), Open Information Model, version 1.1 (Aug 1999).
[44] National Center for Biotechnology Information (NCBI), available in http://www.ncbi.nlm.nih.gov/BLAST/.

[45] W. Paton, R. Stevens, P. Baker, C. Goble, S. Bechhofer, A. Brass, Query Processing in the TAMBIS Bioinformatics Source Integration System, in: Proc. 11th Int. Conf. on Scientific and Statistical Databases (IEEE Press, New York, 1999) 138–147.
[46] D. Rocco, T. Critchlow, Discovery and Classification of Bioinformatics Web Services, Tech. Report UCRL-JC-149963, Lawrence Livermore National Lab, (USA, 2002).
[47] S. Rössle, P. Carvalho, L. Dardenne, P. Bisch, Development of a Computational Environment for Protein Structure Prediction and Functional Analysis, Second Brazilian Workshop on Bioinformatics (WOB 2003), (Rio de Janeiro, 2003).
[48] A. Šali, Modeller: A Program for Protein Structure Modeling Release 6, Rockefeller Univ., (Dec. 2001) available in http://guitar.rockefeller.edu/modeller/modeller.html.
[49] A. Sheth, D. Worah, K. Kochut, J. Miller, K. Zheng, D. Palaniswami, S. Das, The METEOR workflow management system and its use in prototyping significant healthcare applications, in: Proc. 1997 Toward an Electronic Patient Record Conference (TEPR'97), vol. 2, (Nashville, TN, 1997) 267-278.
[50] P. Spyns, R. Meersman, M. Jarrar, Data Modelling versus Ontology Engineering, SIGMOD Record 31 (4) (2002) 12-17.
[51] L. Stein, Creating a Bioinformatics Nation. Nature 417 (2002) 119-120.
[52] S. Thatte, XLANG: Web services for Business Process Design, available in http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, Microsoft Corporation (2001).
[53] VisualScript XML, available in http://www.smartdraw.com/.
[54] W3C (World Wide Web Consortium) Note, Web services Conversation Language (WSCL) 1.0, available in http://www.w3.org/TR/2002/NOTE-wscl10-20020314/ (March 2001).
[55] W3C (World Wide Web Consortium) Note, Web services Description Language (WSDL) 1.1, available in http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (March 2001).
[56] W3C (World Wide Web Consortium) Note on Simple Object Access Protocol (SOAP) 1.1, available in http://www.w3.org/TR/SOAP/ (May 2000).
[57] W3C (World Wide Web Consortium) Recommendation, Extensible Markup Language (XML) 1.0 (Second Edition), available in http://www.w3.org/TR/REC-xml (Oct 2000).
[58] W3C (World Wide Web Consortium), Web Services Architecture, Working Draft (Nov 2002).
[59] Web Services, available in http://www.w3.org/2002/ws/.
[60] Web Services Interoperability Organization, available in http://www.ws-i.org.
[61] G. Wiederhold, Mediation in Information Systems, ACM Computing Survey 27 (2) (1995) 265-267.
[62] M. Wilkinson, M. Links, BioMOBY: An Open source biological web services proposal, Briefings in Informatics 3 (4) (2002) 331-341.
[63] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, L. Moreau, Automating experiments using semantic data on a bioinformatics grid, IEEE Intelligent Systems 19 (1) (2004) 48-55.
[64] C. Wroe, R. Stevens, C. Goble, A. Roberts, M. Greenwood, A suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data, in: Proc. Int. Journal of Cooperative Information Systems 12 (2) (2003) 197-224.

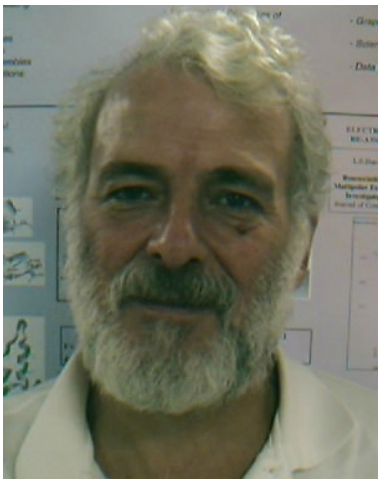
11. Vitae

Maria Cláudia Cavalcanti is a senior system analyst and researcher at NCE-UFRJ. She collaborates with the DCC/UFRJ Computer Science 7Graduation and M.Sc. courses, and participates on research and development projects on Distributed Databases, Metadata Management and Semi-structured data areas. She received a D.Sc. in Systems Engineering and Computer Science from COPPE Sistemas - UFRJ, Brazil.

Rafael Targino received a B.Sc. degree in Computer Science from the Federal University of Rio de Janeiro, Brazil in 2000. He is currently working on his Masters Degree in Computer Science at the same institution. His areas of research include scientifc workflows, web services technology and distributed databases. He works in object oriented software development utilizing agile development processes such as Extreme Programming. He has participated in several projects involving major Brazilian corporations.

Fernanda Baião is a researcher at the Department of Computer Science at the COPPE Institute from Federal University of Rio de Janeiro since 2001, where she participates in the Database and Artificial Intelligence Research Groups. She has received the Doctor of Science degree from Federal University of Rio de Janeiro in 2001. During the year 2000 she worked as a visiting student at the University of Wisconsin, Madison (USA). Her current research interests include distributed and parallel databases, data management issues of web services composition and genome data management, and machine learning techniques. She participates in research projects in those areas, with funding from several Brazilian government agencies, including CNPq, CAPES, FINEP and FAPERJ. She participated in program committees of national and international conferences and workshops, and is a member of ACM and of the Brazilian Computer Society.

Shaila C. Rossle is a PhD. student at the Institute of Biophysics in Federal University of Rio de Janeiro. She participates on research and development projects on Bioinformatics and Structural Biology areas.



Paulo Mascarello Bisch is a researcher and senior professor at the Institute of Biophysics in Federal University of Rio de Janeiro. He was the advisor of 18 PhD thesis in the fields of fluid interface physico-chemistry, membrane biophysics, molecular modeling of biological structures, structural biology and bioinformatics, participating more recently in Genomics and Proteomics cooperative projects. He received a D.Sc. in Sciences, Physics at the Free University of Brussel, Belgium, and was also researcher at the University of Ulm, Germany, at Brazilian Center of Physical Research, Rio de Janeiro and at University of São Paulo, Brazil.

Paulo F. Pires is a Collaborator Professor of the Department of Computer Science at the Mathematics Institute from Federal University of Rio de Janeiro since 2002, where he participates as a researcher of the Dataware Research Group. He has received the Doctor of Science degree from Federal University of Rio de Janeiro in 2002. During the year 2000 he worked as visiting researcher at the CLIP lab in University of Maryland (USA). His main research interests include ontologies, data integration and resource management on the Web, transaction models for the Web environment, infrastructures for Web service composition, and formal modeling tools for distributed systems. Dr. Pires has participated in several cooperative industry-university projects as well as research projects in those areas, with funding from several Brazilian government agencies, including CNPq, CAPES, FINEP and FAPERJ.

Maria Luiza M. Campos is a Professor of the Department of Computer Science at the Mathematics Institute from Federal University of Rio de Janeiro since 1995, where she leads the Dataware Research Group. She has received the Doctor of Philosophy degree from University of East Anglia, United Kingdom. Dr. Campos has participated in several cooperative industry-university projects and has previously worked for 10 years as a Data and Database Administrator at IBGE (Brazilian Institute of Geography and Statistics). Her main research interests include metadata management, ontologies, data integration and resource management on the Web, OLAP and data warehousing. She has coordinated many research projects in those areas, with fundings from the industry and Brazilian government agencies, including CNPq, CAPES, and FAPERJ.

Marta Mattoso is a Professor of the Department of Computer Science at the COPPE Institute from Federal University of Rio de Janeiro since 1994, where she co-leads the Database Research Group. She has received the Doctor of Science degree from Federal University of Rio de Janeiro. Dr. Mattoso has been active in the database research community for more than ten years and her current research interests include distributed and parallel databases, data management aspects of web services composition and genome data management. She is the principal investigator in research projects in those areas, with fundings from several Brazilian government agencies, including CNPq, CAPES, FINEP and FAPERJ. She has served in program committees of international conferences and workshops, and is a reviewer of several journals. She is a member of the Brazilian Computer Society, where she belongs to the steering committee of the database special interest group and ACM, where she is an editor of the ACM-SIGMOD Digital Symposium Collection and a member of the SIGMOD Latin American Liaison Committee.

12. Figure Legends

Figure 1: Web services architecture

Figure 2: The MHOLline workflow definition

Figure 3: The MHOLline Web services architecture

Figure 4: Blast WSDL file

Figure 5: Blast SOAP request

Figure 6: Partners of the MHOLline BPEL4WS process

Figure 7: Containers of the MHOLline BPEL4WS process

Figure 8: BPEL4WS specification for the receive activity

Figure 9: BPEL4WS specification for the assign activity

Figure 10: MHOLline workflow WSDL file

Figure 11: MHOLline workflow definitions: Perl script x BPEL4WS

Figure 12: MHOLline workflow processing

Figure 13: The FASTA input sequence

Figure 14: SRMW architecture

Figure 15: SPMW metamodel

Figure 16: SPMW Program XML Schema

Figure 17: SRMW functionality

Figure 18: Publishing the Blast-P program

13. Tables

Table 1: Experiment execution times (in seconds).

Steps	Web services	Perl Script
Step 1 – MHOLline start → BLAST	0.942	0.023
Step 2 – BLAST → BATS	0.764	0.016
Step 3 – BATS → MODELLER	0.330	0.026
Step 4 – MODELLER → MHOLline end	0.209	0.024
TOTAL	2.245	0.089

Table 2: Execution time of each program of the MHOLline workflow (in seconds).

Programs	Execution Time
BLAST	0.161
BATS	0.095
MODELLER	245.257