


Towards a Scientific Model Management System

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by Infoscience - École polytechnique fédérale de Lausanne

¹ EPFL-IC – Database Laboratory
Lausanne, Switzerland

{fabio.porto, jose.macedo, javiersancheztamargo,
yuanjian.wangzufferey, stefano.spaccapietra}@epfl.ch

² UFC – Department of Computing
Ceará, Fortaleza, Brazil
vvidal@lia.ufc.br

Abstract. Computational models of biological systems aim at accurately simulating in vivo phenomena. They have become a very powerful tool enabling scientists to study complex behavior. A side effect of their success unfortunately exists and is observed as an increasing difficulty in managing data, metadata and a myriad of programs and tools used and produced during a research task. In this work we aim at supporting scientists during a research endeavour by using Scientific Models as a main guiding element for describing, searching and running computational models, as well as managing the corresponding results. We assume a data-oriented perspective for scientific model representation materialized into a data model with which users describe scientific models and corresponding computational models, and a query language with which a scientist specifies simulation queries. The model is grounded in XML and tightly related to domain ontologies, which provide formal domain descriptions and uniform terminology. Scientists may search for scientific models and run simulations that automatically invoke the underlying programs on provided inputs. The results of a simulation may generate complex data that can be queried in the context of the scientific model. Higher-level models can be specified through views that export a unified representation of underlying scientific models.

1 Introduction

Multiple large-scale scientific projects are expected to produce a never before observed amount of experimental and simulation data. Current technology for data management is clearly unable to cope with the needs to store, index, search, analyse, process and integrate such massive, complex and very often-heterogeneous data. Additionally, the very aim of scientific exploration requires a different perspective on data management, one that can cope with expected knowledge evolution and worldwide collaboration.

A promising approach is based on the concept of scientific models [1], which are formal representations synthesizing the understanding about a studied phenomenon, entity or process. It allows scientists to simulate the behaviour of the real world and compare it against experimental results.

From a data perspective, scientific models encompass all the information used and produced during a scientific exploration. It is contextualized by its scientific domain, bibliographic references, a description of the observed phenomenon and, in some cases, some mathematical formulae and provenance information. A computational model, in this paper, refers to a realisation of a scientific model through computational resources, software and hardware. Basically, a scientist or engineer creates a computational implementation that simulates the behaviour of the studied phenomenon and its interaction with interfacing environment, according specifications in the scientific model.

Transforming a scientific model into a computational one can be a complex task, if at all possible, but once realised allows scientists to confront simulation against experimental results, leading eventually to an accurate representation of the studied phenomenon. This process known as model tuning includes both modifying programs (i.e. changing the behaviour) and fitting input parameter and set-up values (i.e. specifying initial simulation state). In many cases, when experimental data are available, regression analysis, provide formal procedures for model fixing [2].

It is a fact that the complexity of specifying and running a computational model and managing all the resources produced during a scientific endeavour deviates the attention off the phenomenon being investigated to implementation concerns [3]. State of the art approaches to computational simulation rescue to scientific workflows languages, such as SCUFL and BPEL, to specify workflows and to their corresponding running environments [4,5, 6,7] for models evaluation. For small scale non data intensive simulations, scientist may choose to use tailored environments like MATLAB [8] and Neuron [9].

Whilst some of these have attracted a large user community [10], they fail to provide an integrated scientific model environment as proposed in this paper, in which the studied phenomenon, the derived models and data therein produced are integrately managed.

In this work we aim at supporting scientists in specifying, running, analysing and sharing scientific models and model's data. To this end we sketch a scientific model management system architecture and detail a data model and query language for specifying scientific models and running simulations. We assume a data-oriented perspective for scientific model representation, materialized into a data model with which users describe scientific models and derived computational models, and a query language with which a scientist specifies simulation queries. The model is grounded in XML and tightly related to domain ontologies, which provide formal domain descriptions and uniform terminology. Scientists may search for scientific models and run simulations that automatically invoke the underlying programs on provided inputs. The results of a simulation may generate complex data that can be queried in the context of the scientific model. Higher-level models can be specified through views that export a unified representation of underlying scientific models.

The remaining of this paper is structured as follows. Section 2 gives a brief introduction to the system architecture. Next, section 3 introduces a running example extracted from the neuroscience domain. Section 4 describes the data model and section 5 presents the simulation query language. Finally, section 6 concludes discussing our achievements so far and future works.

2 Scientific Model Management System

A scientific model management system (SMMS) supports scientists in designing, creating, searching and running scientific models, and in managing the results of simulations and analysis. Figure 1 depicts the main system functions.

The system functions are structured into four layers. A user layer provides the interface for scientists to create and edit elements of the model and to request system services, such as running simulations, querying and reasoning. Users may query scientific model meta-data as well as simulation results.

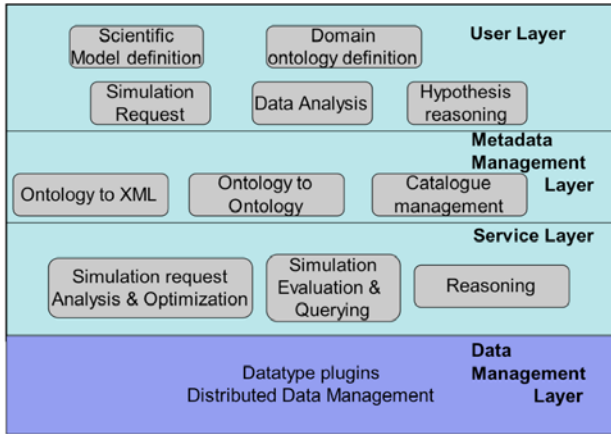


Fig. 1. Scientific Model Management System Architecture

The metadata management layer stores scientific model metadata and supports metadata management services. In this work, scientific model metadata is based on a set of ontologies that guarantees uniform terminology among models and scientists. A transformation and selection service allows scientists to map ontology fragments to XML trees, which are then used in data model elements description. The catalog service manages metadata about scientific model and data, as well as supporting information such as ontologies, views (see section 5.5) and transformation rules.

The service layer supports simulation evaluation, querying of simulation results and reasoning. We have extended the query processing system CoDIMS [11] to cope with simulation queries evaluation.

Finally, a data management layer supports distributed scientific models management and wrappers implementing complex datatypes, offering access to simulation results, such as graphs and time series.

In this paper, the details of the architecture are not further explored. Instead, the following sections present the backbone of the system in the form of a data model and simulation language. Similarly, the details regarding ontology managing, transformation and alignment are left to future work.

3 A Neuroscience Scientific Model

In this section we introduce a running example taken from scientific models developed for the neuroscience domain. According to Kandel [12], “the task of neural science is to explain behavior in terms of the activities of the brain”. Numerous computational neuroscience groups investigate scientific models that aim at explaining such behavior. A classical example is the axon membrane action potential model proposed by Hodgkin and Huxley (HH) [13] that describes how action potentials traverse the cell membrane. An action potential is a pulse-like wave of voltage that can traverse certain types of cell membranes, such as the membrane of the axon of a neuron. The model quantitatively computes the membrane potential between the interior and the extracellular liquid, based on the flow of ions of sodium (Na⁺), potassium (K⁺) and a leakage one representing all other channel types. The mathematical equation proposed by HH is presented below:

$$I = m^3 h g_{Na} (E - E_{Na}) + n^4 g_K (E - E_K) + g_L (E - E_L) \quad (1),$$

where g_i , $i=\{Na, K, L\}$, are function of time dependent variables, representing the membrane conductance, and E_i model the equilibrium potential for each ion channel, while E is the membrane potential. Finally, n , m and h are parameters controlling the probability of the sodium or potassium gates to be opened. The total ionic current across the membrane of the cell is modeled by the variable I .

The action potential model defined by HH simulates the variation on voltage in the cell when applied to a neuron compartment. From a conceptual point of view, one can represent a single neuron model with its various compartments and the membrane behavior given by the HH model by the ontology in Figure 2. The Hodgkin-Huxley class models the corresponding model with the input and output parameters conforming to equation (1).

From the scientific model specification, a computational neuroscientist will conceive programs that implement the behavior defined by equation 1. Next, when running a simulation, the program receives input values for the parameters identified in (1) and produces the total ionic current across the membrane (variable I in (1)).

In the next section, the HH model is used as a running example illustrating the scientific model data model elements.

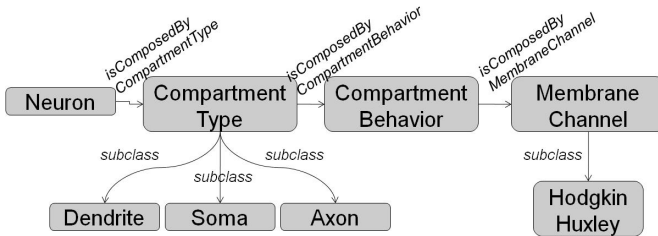


Fig. 2. Single Neuron with Hodgkin-Huxley model domain ontology

4 The Scientific Model Data Model

The backbone of the SMMS is a data oriented semantically based data model and simulation language. During a scientific exploration, a scientist registers data and metadata describing the scientific and corresponding computational models. Scientific model metadata provides provenance, contextual and descriptive information aiding on model search and querying. Similarly, computational model metadata is used as the basis for the automatic evaluation of simulations and serves as context for the input and output data. Thus, the data model identifies the following main composing elements: the scientific model, the computational model and simulations. The following subsections detail each one of these layers.

4.1 The Scientific Model Layer

The first layer describes a Scientific Model (SM). It accompanies the first step in an experiment or simulation specification, in which scientists describe the research problem being pursued. It considers the scientific domain, setting the context to the problem, and the formal problem specification in the form of mathematical formulae, if adequate. A formal description of a SM is given in (1) and includes: a resource definition R , a list of bibliographic references (B), possibly images (I), and annotations (A). R is a resource description defined as a tuple $\langle \text{LSID}, D \rangle$. LSID is an unified identifier for the life science specified as $\text{urn:lsid:authority.org:namespace:object:revision}$, where urn:lsid is the LSID protocol identifying label, authority.org is a DNS reference, and the remaining field names are self explanatory [14]. D is a free text resource description. We use domain ontologies to formally describe the domain covered by the scientific model (O_{SMD}) and to explain the mathematical formulae (O_{MF}). The LSID identification attribute hooks the scientific model to its computational models and simulations.

Thus, a scientific model is defined as a 6-tuple $\text{SM} = \langle R, O_{SMD}, O_{MF}, B, I, A \rangle$ (1).

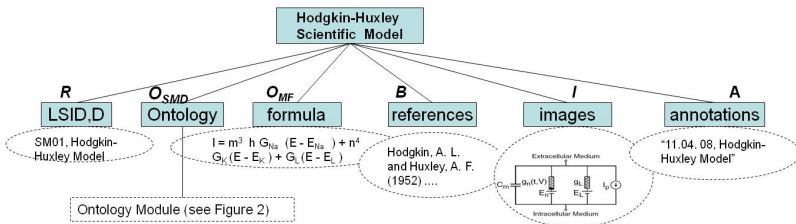


Fig. 3. The HH scientific model representation

The HH model, presented in section 3, can be depicted as a scientific model. Its data view is illustrated in Figure 3. It provides metadata to support basic search queries over a scientific model database and reasoning capabilities on the theory specified by the scientific model ontology.

4.2 The Computational Model Layer

A Computational Model (CM) realises a Scientific Model through a computational implementation. From the point the view of the research task, a scientist develops software that reproduces the behavior specified by the scientific model. The objective of the computational element is to describe its components and provide sufficient metadata to allow automatic instantiation of a CM when requested by a simulation query (see section 5.4). The computational model may comprehend a series of orchestrated programs whose scheduling is hidden by an exposed façade entry point. Indeed, a computational model is the basis for a simulation query that provides values matching CM input requirements to produce the simulation output.

A CM specification comprehends two ontologies: environment and domain. The former describes the execution environment associated to the CM, including: programming language specification, programming environment parameters, documentation, input and output parameters, initialization procedure, and executing programs. The set of input parameters expected by a CM can be divided into two groups: set-up and input parameters. The CM set-up parameters include state information that should hold during various simulations. Set-up parameters may also introduce run-time control values, such as frequency of output graph update etc. The set-up parameter are referred to in the data model as $I_S = \{i_1, i_2, \dots, i_n\}$, where each instance $i_j = \langle v, P \rangle$ of I_S corresponds to a value v and its corresponding parameter P in Parameters.

Additionally, for each computational model, an *outputWrapper* specifies methods for obtaining its results. Given the different formats and procedures used by generic programs when producing output, each CM shall indicate the *outputWrapper* class that knows how to capture its output and return it to the system to feed an eventual pipeline.

The CM domain ontology contextualizes CM parameters according to the O_{SMD} ontology. Contextualization is obtained by associating each parameter to its semantic meaning as an instance of a ontology concept. In fact, we propose a transformation of the domain ontology into a XML serialization tree, in which nodes correspond to ontology concepts, and edges, from parent to children are either ontology roles, having the role domain being the parent node in XML, or is-a relationships. Thus, as shown in Figure 4, the output variable “I” is mapped to the XML path “Neuron/Axon/Hodgkin-Huxley/I”, which in turn corresponds to a path in the ontology specified in DL-SWRL [15] as:

$$\begin{aligned} & \text{Neuron}(?x) \wedge \text{isComposedByCompartmentType}(?x, ?y) \wedge \text{Axon}(?y) \wedge \\ & \text{isComposedByCompartmentBehavior}(?y, ?z) \wedge \text{CompartmentBehavior}(?z) \wedge \\ & \text{isComposedByMembraneChannel}(?z, ?a) \wedge \text{HodgkinHuxley}(?a) \wedge \\ & \text{hasTotalIonicCurrent}(?a, ?I) \rightarrow \text{TotalIonicCurrent}(?I) \end{aligned}$$

A CM model is formally defined as a 7-tuple $\text{CM} = \langle R, \text{LSID}_{SM}, \text{XO}_E, \text{XO}_D, \text{M}_i, \text{M}_o, A \rangle$ (2).

In (2), R is the CM resource identification; LSID_{SM} is a reference to the associated scientific model, XO_E and XO_D , are the XML serializations of the environment ontology, of XML type Environment, and of the domain ontology, of XML type Neuron, respectively. In addition, M_i (M_o) defines mappings between the underlying program

input (output) parameters and corresponding domain ontology properties (XML tree leave node). Finally, *A* corresponds to annotations identifying authoring information.

Figure 4 illustrates the representation of a computational model implementing the scientific model SM01.

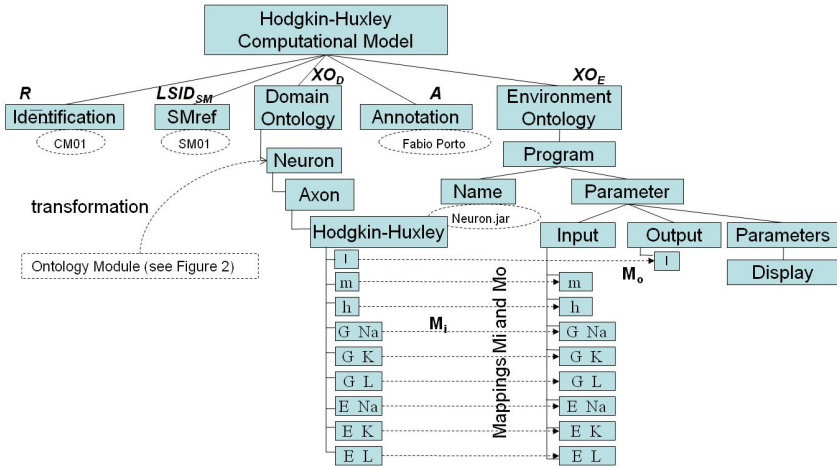


Fig. 4. A Hodgkin-Huxley Computational Model

4.3 Simulation Layer

The third layer in the scientific model data model specifies a simulation query (*S*). Whilst the two first layers present metadata about a scientific model and its computational implementation, a simulation query combines CMs into an evaluation. A simulation query specification provides the initial state of the simulation through its input parameter values and a particular set-up. Section 5 will describe the simulation query language.

5 Simulation Language

Syntactically, a simulation query is specified as an expression comprising a head and a body. The body is a boolean expression composed of a conjunction of simulation query predicates, whereas the head lists variables containing the expected simulation results, necessarily appearing in one of the simulation predicates in the body. We start by presenting the syntax and semantics of simulation query predicates.

5.1 Simulation Predicate

A simulation query predicate is specified as:

$$S_i((V_i, W_i) ; (X_i', X_o') ; (I_i, O_i) ; I_s) \quad (3).$$

In (3), S_i labels the simulation query predicate according to the CM resource identification. In order to easy references to the computational model input and output values, two sets of variables are defined, respectively, V_i and W_i . These variables refer to values provided as input or produced as output when running the underlying CM. Indeed, users interface with simulations queries by providing input parameter and set-up values, and by receiving back the output values computed as a function of these inputs. The set of input and output parameters' values are provided by the XML documents X_i' and X_o' , respectively, whereas I_s represents simulation set-up parameters. Note that the associated CM definition specifies the schemas for X_i' and X_o' . For example, using the CM example in Figure 4, the X_i' document can be obtained from the result of the XPath expression “/CM/DomainOntology” over the HH CM XML definition, and by filling its leaf nodes with the input values. Thus, /Neuros/Axon/Hogking-Huxley/m = 0,1, illustrates a possible value assignment for the input parameter m . Finally, the mappings $I_i(O_o)$ define the correspondence between the input-output variables in V_i and W_i and the input-output parameter values in X_i' and X_o' .

Definition 1. correspondence assertions in I_i and O_i are specified as $\$x = Path$, where $\$x$ is a variable in $\{V_i \cup W_i\}$ and $Path$ is an XPath [16] expression pointing to a data element in X_k' , $k=\{i,o\}$, whose child node is either an input parameter value or an output value.

Having described the syntax of individual simulation predicates, the semantics of body expressions can be announced. Initially, the semantics of a single simulation predicate is exposed followed by one for complete body expressions.

5.2 Semantics of a Single Simulation Predicate

A single simulation predicate returns a boolean value of its evaluation according to the definition in Def 2 below with respect to the its syntax in (3) and the CM specification in (2).

Definition 2. A simulation predicate S_i is assumed to be *true* iff given a X_i' holding the set of input parameter values to the program implementing the corresponding CM, according to M_i , it exists a X_o' whose leaf values are produced by the evaluation of the referred program and that is built from the mappings in M_o .

5.3 Semantics of the Body of a Simulation Expression

Once the semantics of a single simulation predicate is specified as a boolean expression, more elaborate conjunctive expressions can be composed to form the body of a simulation. The semantics of a conjunction of simulation predicates in the body of a simulation is defined in Def 3.

Definition 3. Given a conjunction of simulation predicates $s = s_1 \wedge s_2 \wedge \dots \wedge s_n$, s is considered to hold *true* if the conjunctive expression on the right evaluates to true. Moreover, if more than one simulation predicates s_i and s_j in s , refer to the same variable, for instance $\$x$, then they share a single associated value. In addition, the shared variable must hold a single binding to a value, either provided as input or produced as output by an underlying program computation.

Note that the restriction regarding sharing variables among simulation predicates leads to data-dependency relationships, in which the simulation predicate holding the associated value to the shared variable shall precede in evaluation order the remaining simulation predicates sharing that particular variable. Moreover, variable sharing introduces a particular mode of value assignment to data elements in X_i' , replacing that of the node corresponding to its associated path.

Finally, given a body that evaluates *true*, then the head of the simulation identifies the variables in the simulation predicates whose values are returned as the simulation outputs, such that if K is the set of variables in the head then

$K \subseteq (V_i \cup W_i)$, for $1 \leq i \leq n$, with n being the number of the simulation predicates in the body.

5.4 A Simulation Query

A simulation query combines the head and its body into a simulation clause as illustrated in (4), according to Def 1, 2 and 3.

$$S(K) = S_1((V_1, W_1); (X_{i1}', X_{o1}'); (I_1, O_1); I_{S1}) \wedge \\ S_2((V_2, W_2); (X_{i2}', X_{o2}'); (I_2, O_2); I_{S2}) \wedge \dots \wedge \\ S_n((V_n, W_n); (X_{in}', X_{on}'); (I_n, O_n); I_{Sn}) \quad (4)$$

An example of a simulation query is depicted in Figure 5. This particular query returns the total ionic current across the membrane ($\$I$) according to the parameters values specified in the input document $HHCM01_i$. As discussed before, the user must provide a mapping from each query variable to the corresponding data element of the domain ontology XML serialization document. In this example, the input and output XML documents, X_i' and X_o' , are illustrated by documents $HHCM01_i$ and $HHCM01_o$, respectively, both of type Neuron.

```
S($I, $Z) = CM01(($m,$h,$G_Na,$G_K,$G_L,$E_Na,$E_K,$E_L,$I);
    (HHCM01i, HHCM01o);
    ($m = /Neuron/Axon/ Hodgkin-Huxley/m,
    ...1,
    $I = /Neuron/Axon/ Hodgkin-Huxley/I)) ^
    CM022 (($I, $Z); (ACM02i, ACM02o);
    ($Z=/Analysis/result))
```

Fig. 5. Simulation query example

5.5 Simulation View

One may want to register a simulation so that it can be re-executed later on or included in a more complex simulation. A registered simulation is called a view, borrowing the term from database literature [17], as it provides users with an external

¹ The remaining mappings are not shown due to lack of space.

² The CM02 computational model has purposely not been described.

perspective of a simulation through the set of input parameter values that configure the computational models taking part into the simulation view. In addition, a simulation view establishes correspondences between the exported parameters and the ones specified on each simulation predicate taking part in the body of the simulation describing the view. In (5) a simulation view is depicted:

$$\begin{aligned}
 S_v((V, W); (X_{iv}', X_{ov}'); (I_v, O_v); (I_{Sv}, M_s)) = \\
 S_1((V_1, W_1); (X_{i1}', X_{o1}'); (I_1, O_1); I_{S1}) \wedge \\
 S_2((V_2, W_2); (X_{i2}', X_{o2}'); (I_2, O_2); I_{S2}) \wedge \quad (5) \\
 \dots \wedge \\
 S_n((V_n, W_n); (X_{in}', X_{on}'); (I_n, O_n); I_{Sn}).
 \end{aligned}$$

The body of the simulation view reflects the one in ordinary simulations, expressing conjunction of simulation predicates. The difference appears in the head of the formulae. Indeed, the latter exports an integrated view of the simulation predicates' input and output parameters appearing in the body of the formulae and specified in X_{ik}' and X_{ok}' , $1 \leq k \leq n$. The two sets of correspondences, I_k and O_k , map the external view in $\{X_{iv}', X_{ov}'\}$ to the corresponding parameters in the simulation predicates in the body, $\{X_{ik}', X_{ok}'\}$. Thus, a correspondence assertion is expressed as $S_v.path/dataelement \equiv S_i.path/dataelement$, where path is an XPath expression. In the same line as the input/output parameters, I_{Sv} expresses the uniform view of set-up parameter values appearing in the body of the formulae and M_s asserts the correspondences between the set-up data elements in I_{Sv} and those in the body.

6 Conclusion

Managing *in silico* simulations have become a major challenge for eScience applications. As science increasingly depends on computational resources to aid solving extremely complex questions, it becomes paramount to offer scientist mechanisms to manage the wealth of knowledge produced during a scientific endeavor. This paper presents initial results aiming to contribute to this idea. We propose a data centric semantic based data model with which scientists represent scientific models and associated computational models. Furthermore, the data model supports specifying and running simulations as a function of computational models over inputs. The intention of this work is to provide a basic backbone from which more complex services can be developed as their needs come. We have developed a first prototype system that implements the data model and the simulation query language on top of the CoDIMS system. A first scientific model management system architecture is proposed with a set of minimal services that scientists may expect from such an environment.

There are many future works already being explored. Indeed, the investigation of scientific hypothesis may introduce uncertainty to the data model. Another problem is related to the format and semantic mismatch between computational models data in a simulation. We are interested in studying how the results on heterogeneous databases and ontologies can be applied to the scenario discussed here. Finally, the integration of simulation results with knowledge in ontologies and the exploration of their content as complex data structures within a query language is also a subject of future research.

References

- [1] Hunter, J.: Scientific Models – A User-oriented Approach to the Integration of Scientific Data and Digital Libraries. In: VALA 2006, Melbourne (February 2006)
- [2] Jaqaman, K., Danuser, G.: Linking data to models: data regression. *Nature Reviews, Molecular Cell Biology* V(7), 813–819 (2006)
- [3] Silvert, W.: Modelling as a discipline. *International Journal General Systems* V30(3), 1–22 (2000)
- [4] Oinn, T., Greenwood, M., Addis, M.: Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrence Computation: Pract. Exper.*, 1–7 (2000)
- [5] Akram, A., Meredith, D., Allan, R.: Evaluation of BPEL for Scientific Workflows. *Cluster Computing and the Grid, CCGRID V1(16-19)* (2006)
- [6] <http://www.gridworkflow.org/snips/gridworkflow/space/XScufl>
- [7] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Kepler, M.: An Extensible System for Design and Execution of Scientific Workflows. In: SSDBM (2004)
- [8] MATLAB (last access, 24/06/2008), <http://en.wikipedia.org/wiki/Matlab>
- [9] Neuron (last access, 24/06/2008), <http://www.neuron.yale.edu>
- [10] Roure, D., Goble, C., Stevens, R.: Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows. In: e-Science 2007 - Third IEEE Int. Conf. on e-Science and Grid Computing, Bangalore, India, December 10-13, 2007, pp. 603–610 (2007)
- [11] Porto, F., Tajmouati, O., Silva, V., Schulze, B., Ayres, F.: QEF - Supporting Complex Query Applications. In: CCGRID 2007, Rio de Janeiro, Brazil, pp. 846–851 (2007)
- [12] Kandel, E., Schwartz, J., Jessel, T.: *Principles of NeuroScience*, 4th edn. McGraw-Hill, New York (2000)
- [13] Hodgkin, A., Huxley, A.: A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. *J. Physiol (Lond.)* 117, 500–544 (1952)
- [14] (last accessed, 26/04/2008), <http://lsids.sourceforge.net/>
- [15] Grosz, B., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: Proc. WWW 2003, Budapest (May 2003)
- [16] (last accessed, 26/04/2008), <http://www.w3.org/TR/xpath>
- [17] Elmasri, R., Navathe, S.: *Fundamentals of Database Systems*, 2nd edn. Benjamin/Cummings (1994)