Design and Architecture of Web Services for Simulation of Biochemical Systems

Joseph O. Dada[1] and Pedro Mendes[1,2]

[1] School of Computer Science and Manchester Centre for Integrative Systems Biology, The University of Manchester, Manchester Interdisciplinary Biocentre, 131 Princess Street, Manchester M1 7DN, UK
[2] Virginia Bioinformatics Institute, Virginia Tech, Washington Street 0477, Blacksburg, VA 24060, USA
{joseph.dada,pedro.mendes}@manchester.ac.uk

Abstract. Computer simulation of biochemical networks plays a central role in systems biology. While there are several software packages for modeling and simulation of these networks, they are based on graphical user interfaces that operate on the local computer. However, it is now often desirable to operate simulation tasks through a distributed computing framework where data can be gathered from different sources and distinct subtasks operated in different physical computers. In this paper we describe a web services implementation for the COPASI biochemical network simulator (CopasiWS). COPASI provides a range of numerical methods for simulation, optimization and analysis of biochemical reaction networks. Our aim is to allow easy integration of these powerful functionalities with local and remote services to provide a distributed computing platform for the simulation and analysis of biochemical models. One immediate result of this work is that simulation tasks are now available to be used in a platform– and language–independent manner as components of distributed workflows, for example using the Taverna workflow engine. We describe the CopasiWS architecture, key design and implementation issues, and illustrate the range of services available through a web portal interface (CopasiWeb).

1 Introduction

Systems biology is a new approach to biological research where experimental design and analysis are carried out based on a view of cells as systems of interacting components, rather than each of its molecular components in isolation. A major part of this new approach is fulfilled by mathematical and computer models of the underlying interaction network which are used for simulation. There are several software packages for modeling and simulation of these networks, such as COPASI [1], or CellDesigner [2], which are mostly based on graphical user interfaces that operate on the local computer. The field of biochemical simulation has advanced mostly in terms of the analytic frameworks and numerical algorithms, but not so much in terms of architecture, which is either based

N.W. Paton, P. Missier, and C. Hedeler (Eds.): DILS 2009, LNBI 5647, pp. 182–195, 2009.

on single–computer or client–server solutions. Recent advances in grid and distributed computing suggest that it would be desirable to operate modeling and simulation of biochemical systems in a language independent manner and in a distributed computing environment. Such a system would allow accessing data from different sources and carrying out different computational operations in different servers on the network resulting in distributed simulations.

An earlier solution that allows for interoperable modeling and simulation tasks is the Systems Biology Workbench (SBW) [3]. SBW is based on a communication bus and a simple messaging system that allows applications to interoperate; SBW contains several applications that carry out specialized operations, from computational analysis to visualization. Because the SBW messaging bus is based on sockets, it is possible to run tasks in different computers and thus this system already allowed for distributed computing. However, SBW is based on a non–standard messaging protocol which is adopted only by a few specialized applications. Unfortunately this means that non–systems biology applications are not able to link with the SBW messaging system and thus this pioneering effort is unlikely to fulfill the full potential of distributed computing.

Web services are a widely used standard for interoperable machine–to–machine interaction over a network. Web services follow standard protocols approved by the W3C organization. A Web service is a system implemented by a software agent capable of sending and receiving messages defined through the Web services description language (WSDL, an XML–based language), and passed through the HTTP protocol. WSDL makes possible the discovery of web services through service brokers.

The advantage of using Web services to facilitate distributed computing is that unlike the case with SBW many more applications are immediately available, even outside the realm of systems biology. For example most bioinformatic resources are accessible through Web services (e.g. the KEGG [4], NCBI [5] and EBI databases [6], as well as sequence analysis tools). There are also many other scientific resources and even commercial services (e.g. Amazon.com). More specific to systems biology, the Biomodels database [7], MIRIAM [8], and the Systems Biology Ontology [9] are accessible through Web services. Workflow Management Systems such as Taverna [10] and Triana [11] allow programs to be created that combine these Web services in arbitrarily complex workflows – this allows the output of an application to become the input to another, resulting in computations carried out across the network, each task by specialized providers.

The aim of this paper is to describe a Web services implementation of the COPASI biochemical network simulator (hereafter named CopasiWS). COPASI [1] provides a range of numerical methods for simulation, optimization and analysis of biochemical reaction networks. COPASI, which is the successor to Gepasi [12], is available for all common operating system platforms and is currently supplied in two versions: one based on a graphical user interface (CopasiUI) and one that is entirely driven through the command line (CopasiSE). CopasiSE was designed such that simulations can be processed in batch mode without user intervention. COPASI is able to read the SBML standard format for systems biology network

models [13], but also has its own file format (CopasiML, also based on XML). While SBML only encodes a model, CopasiML also encodes all of the operations to be carried out with the model and definitions of report files containing output from the simulations.

Providing a Web services interface to COPASI will allow usage of simulation and modeling tasks in distributed workflows, and it makes possible the creation of simulation servers where computing cycles could be offered. Because COPASI carries out a wide range of analyses the result of this work is a large set of specific modeling and simulation Web services. The rest of this paper describes the design, architecture, and prototype implementation of CopasiWS. It ends with an illustration of the range of available services using a simple Web portal interface. Finally a discussion is presented of the perspectives that Web services can bring to systems biology.

2 Design of COPASI Modeling and Simulation Web Services (CopasiWS)

Software clients interact with Web services through interfaces to use the functionality provided by the service. In order to fulfill the client request, the Web service interacts with the appropriate business logic, which may also need to communicate with the application resource to accomplish the client request. The flow of information to accomplish a client request in CopasiWS can be modeled abstractly as a three–layer architecture as shown in Fig. 1. The design and implementation of CopasiWS need to take into consideration the various components that will reside in each of the layers. In this section, we describe the issues that had to be considered in the design of CopasiWS and details of these layers.

2.1 CopasiWS Design Issues

Web Service Interface Development Approach. A top–down approach was used to develop the interface to avoid interoperability problems and to conform with WS–I [14]. Basically the development starts with the definition of the WSDL. A stub code for the service is generated automatically and the developer then completes the implementation of the Web service.

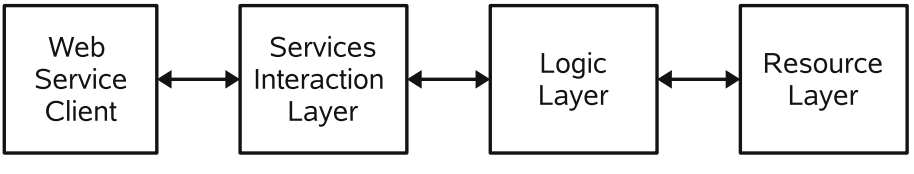

Fig. 1. A three–layer architecture model of CopasiWS. It shows a typical flow of information to accomplish client request to CopasiWS.

Service Granularity. One of the main issues in building Web services is the identification of the right granularity for the services provided. The COPASI user interface defines independent tasks and thus it was natural to follow this granularity for CopasiWS. Each COPASI task has specific input parameters and generates output in predefined text files (many are tab–delimited) or alternatively in some user–defined format. Some tasks have quite simple interfaces, while others are complex and require many parameters to be defined. For example, the time course simulation task can be carried out with deterministic, stochastic or hybrid algorithms, each one requiring a set of specific control parameters. An alternative to providing each task as a single service would be to process the entire specifications of a CopasiML document, like what is done by the command–line batch tool CopasiSE. Providing each of the tasks as a single Web service facilitates composing high–level workflows on top of these basic services. This has the further advantage that since each service only runs one task, it becomes natural to pass the model as an SBML model because then the remaining inputs are only a few other parameters. This is advantageous because otherwise client software would require being able to understand the CopasiML syntax and semantics (CopasiML is specific for COPASI while SBML is understood by numerous packages and a library is available [15] to manipulate files encoded in SBML).

Synchronous versus Asynchronous Web Services. Some COPASI tasks can take very short runs (seconds or less), however others can be very long (hours or more). Task execution time is usually proportional to the size of a model, but even for a single model different tasks are much slower than others (e.g. a single steady state calculation takes a lot less time than a fine–grained scan of parameter space). Usually tasks that take short computing times are better operated in a synchronous mode, where the client blocks waiting for the final results from the service. Others that are long are better operated in an asynchronous mode where the client first requests the service then checks at discrete times whether the service has ended before retrieving results. Unfortunately it is impossible to predict the length of a task and thus it is not possible to select synchronous vs. asyncronous modes automatically; the requestor of the service must decide a priori which mode of operation to use. To address these two modes of operation each of the tasks are implemented by one synchronous and one asynchronous Web service.

The synchronous implementation is straightforward since it consists of simply creating an intermediate CopasiML input file, running CopasiSE and returning the results in an appropriate format. On the other hand, asynchronous services could be implemented through various patterns of operation. We choose request/reply operations with a polling pattern that consist of many synchronous operations in which the clients initiate all the interactions (i.e. no call–back functions are needed). In this approach, the client is issued an identifier in the first call to the service, and this is then used for subsequent operation calls. This approach decouples the client and service provider and make the implementation of client applications very simple.

Parameter Types of Web Service Operations. A client calls an appropriate method with parameters on the service interface to use the functionality of the service. The parameter types determine the style of the interface. Parameters may be passed as either JAX–RPC value types or XML documents. CopasiWS uses the XML document mode of parameter passing because some of the COPASI tasks require a complex set of parameters.

COPASI Task Input. In order to maximize the level of compatibility of CopasiWS with other systems biology software tools, it is important that it can take SBML files as input. This means that a series of additional parameter values must to be passed as input, since they are needed for the operation of the task but are not part of the SBML specification. Alternatively, the CopasiML format contains all required parameters and therefore we also defined versions of the services that take this format as input. The latter mode suggests that the CopasiUI GUI–based software itself could be a client of CopasiWS; this would allow users to chose between running tasks locally or remotely. The SBML input versions of the services are better suited for workflows that use other SBML software, such as the Biomodels database or the a wrapper to libSBML [28]. The CopasiWS suite also includes a service to convert between the SBML and CopasiML formats. When clients use the SBML version of services, the system translates the SBML into CopasiML, then sets the appropriate task and associated parameters in the CopasiML document and then calls CopasiSE to run the task.

COPASI Task Output. COPASI results can be formatted in flexible ways, but there are already some predefined tab–delimited data formats for specific tasks. These, however, are not very useful for Web services and make further processing of the output data by the clients difficult. It is usual to have Web services results be encoded in XML so we decided to format task results in the Systems Biology Results Markup Language (SBRML), which is an XML-based language to encode systems biology simulation results and experimental data [16].

Many modeling tasks are oriented towards changing details of a model rather than producing numerical results. The results of these tasks are then better suited to be expressed directly in SBML or CopasiML. The Web services that encapsulate those tasks then produce output in exactly the same format as their input. An example of such tasks is parameter estimation, which takes a model together with some data and changes some numeric parameters of the model to best match the data; the result being a new version of the model.

Experimental data. COPASI's parameter estimation task has a more complex input than other tasks as it requires not only a model but also target data that the model should fit. COPASI takes these data in a tab–delimited format where columns of data correspond to some model entities. The data columns must then be mapped to model entities, a tiresome task that is usually done through the GUI by the user. However the purpose of the SBRML format mentioned above [16] is exactly to encapsulate data that is mapped to a model and so it is the most appropriate means to pass these data to the parameter estimation Web

service. The current implementation of this Web service decodes the SBRML and converts it to COPASI's native format; later it is expected that COPASI itself be able to read SBRML directly, which will simplify the present code.

Simulation Resource Management. To process long simulation tasks CopasiWS provides asynchronous processing. For each asynchronous simulation request the system creates a simulation resource. A simulation resource is identified by a unique identifier and has associated a number of files which are stored transiently in the server (the SBML or CopasiML files, simulation output file, experimental data file, resource life time, etc.) CopasiWS therefore needed a mechanisms to manage the resources created by the users.

Simulation Engine Management. CopasiSE, the command line driven version of COPASI, is the ultimate executor of the actual tasks and resides in the resource layer (see Fig. 1). Because CopasiWS is available to many potential clients simultaneously, there will be at times the need to access several instances of CopasiSE simultaneously. This calls for having high–performance or high–throughput computing resources in the server, and this also requires specific modules to manage these resources. As an example we have implemented a high-throughput simulation engine module using the University of Wisconsin's Condor system [17]. Condor creates pools of computers which make their computational power available when idle. In our prototype, CopasiSE jobs are queued to a Condor pool whenever needed by asynchronous Web service calls. Of course, for a responsive and fast service one should set aside sufficient dedicated machines to deal with the required throughput (though these could also be managed through the Condor system if needed).

2.2 Design of CopasiWS Interface

Most of the efforts in designing a Web service interface are spent on the service operations. In this case there was a clear guideline consisting in the COPASI user interface since the Web services follow the same division of tasks. Here we briefly describe the functionalities of each task.

Importing/Exporting and Validation Tasks. COPASI provides methods for converting files from SBML to CopasiML format and vice versa, and also to validate files in either of these formats. These tasks are exposed as Model Processor Web service with operations for converting between SBML and CopasiML, and also for XML schema validation.

Steady State Task. Systems biology models are dynamic models which may be able to reach steady states (*i.e.* where the values of their variables do not change). This task finds one steady state of the dynamical system if it exists. The numerical method used in this task has a number of control parameters, including a tolerance value, and which numerical methods to solve for the steady state (Newton–Raphson and/or numerical integration).

Time Course Task. Since these models are dynamical systems, one of the most basic tasks is to determine a trajectory of the system given an initial condition (which is specified in the input files). COPASI provides different methods for calculating the trajectory: *Deterministic* (which uses the LSODA ordinary differential equation solver [18], *Stochastic* (using Gillespie's stochastic simulation algorithm [19] and *Hybrid simulation* (using a combination of the previous two [1]). Each of these methods has a different set of control parameters. All of these tasks are available through a single Time Course Web service (which can use any one of the three methods).

Metabolic Control Analysis (MCA) Task. MCA is a special type of sensitivity analysis which has a particular interpretation for metabolic networks [20] and quantifies how much the rates of reactions affect the concentrations or fluxes at the steady state. This task has only one control parameter that dictates whether the coefficients are to be scaled or unscaled.

Optimization Task. COPASI provides a framework to find optima of any model state variable or any arbitrary function of state variables. It does this by providing a series of alternative numerical optimization methods, from traditional gradient–based to stochastic global optimizers. Each of these algorithms requires its own particular control parameters. The interface of the Optimization Web service allows the calling function to choose the optimization method and the objective function.

Parameter Estimation Task. Biochemical models depend on many numerical parameters, but quite frequently their values are unknown and have to be estimated from some data set [21]. This task makes use of the optimization algorithms mentioned in the previous section to minimize a least squares function (representing the distance between the model simulation and a set of experimental data). Because this task usually requires high computational demands, we have implemented it as an asynchronous Web service only. The client of this service needs to first call the *CreateSimulationResource* operation, which creates a simulation resource with a unique id which is returned to the client. The client then uses this resource id as one of the parameters for the subsequent operation calls. There are operations for submitting the model, for submitting the experimental data, for starting the simulator and for checking and collecting the results. The result of this Web service is a model in SBML or CopasiML formats, plus a set of statistics of the goodness of fit.

2.3 CopasiWS WSDL Development

We used the functionalities of the COPASI tasks described above to develop a WSDL specification for CopasiWS. Operations are grouped into port types, which describe abstract end points of each Web service. The message element of WSDL defines the data elements of an operation. The message data types are defined using a platform neutral XML Schema syntax. Because there are some data types that are common to some of the Web services, we created a separate

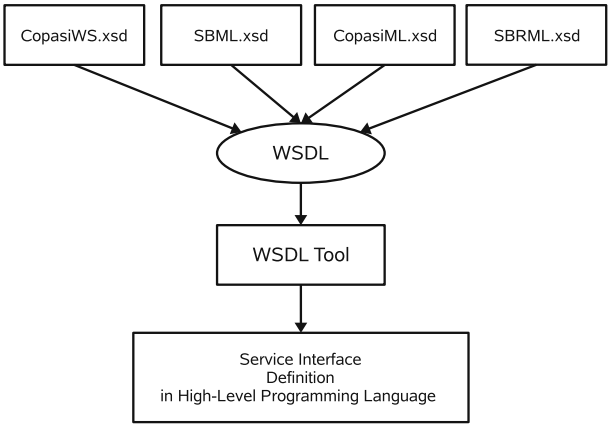

Fig. 2. CopasiWS WSDL development process

XML Schema for the data types (CopasiWS.xsd). We use these data types together with other XML Schema data types for biochemical models (SBML.xsd and CopasiML.xsd)and simulation results (SBRML.xsd) to develop the WSDL for the Web services. Appropriate WSDL toolkit was used to generate the service interface in a specific programming language as represented in Fig. 2.

3 CopasiWS Architecture

Figure 3 depicts the overall architecture of CopasiWS. The three layers already depicted in Fig. 1 are shown here with their internal components. The client communicates with the Web services interaction layer using standard Web service calls, while the interaction layer communicates with the logic layer using a local proprietary interface mechanisms. The logic layer again communicates with the resource layer in order to accomplish the client requests. This model provides a loose coupling between layers and makes it easy to implement components of each layer independently. It also allows for changes in the internal layers while keeping the same interface to the outside world.

3.1 Resource Layer

This layer contains the simulation engine (CopasiSE) that runs the simulation tasks submitted by the logic layer. It also contains a database server that holds relevant information about users and the simulation results.

3.2 Logic Layer

This layer contains various components that help the services interaction layer to accomplish the client request. It contains the following components: simulation

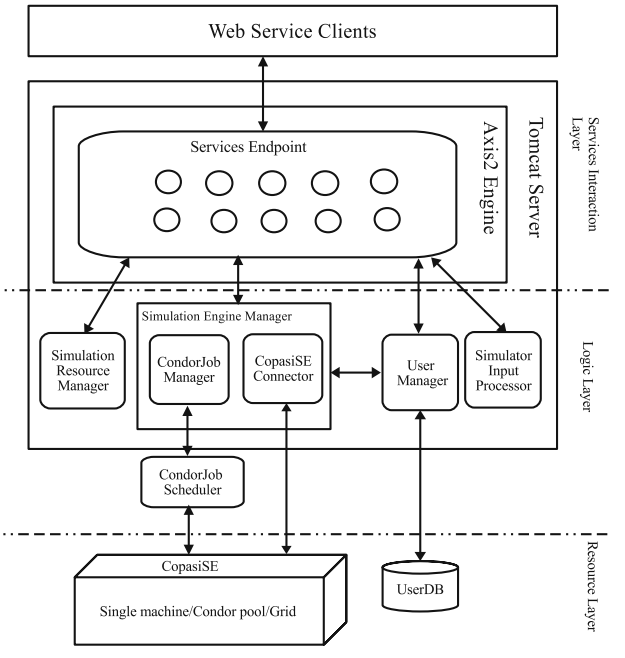

Fig. 3. CopasiWS Architecture

resource manager, user manager, simulator input processor, simulation engine manager and the Condor job scheduler [17], which is a third party component (other grid queue managers could be implemented here as alternatives). The components have well–defined interfaces for communicating with the services interaction layer.

3.3 Services Interaction Layer

This layer provides the glue between the clients and the service functionalities. All the Web service interfaces reside in this layer, and it is responsible for starting the processes in the logic layer in response to client requests. It is also responsible for routing requests to the appropriate components of the logic layer.

The communication between a service endpoint in the interaction layer and the components of the logic layer is determined by the type of request from the clients. An example of how the components of the layers send messages to one another to accomplish a client request (e.g. *runSimulator* operation call to a service endpoint) is shown in Fig. 4.

4 Prototype Implementation, Testing and Example Usage

We have implemented a prototype of the CopasiWS. The services interaction layer is developed in Java programing language [22]. We used an Apache Axis 2

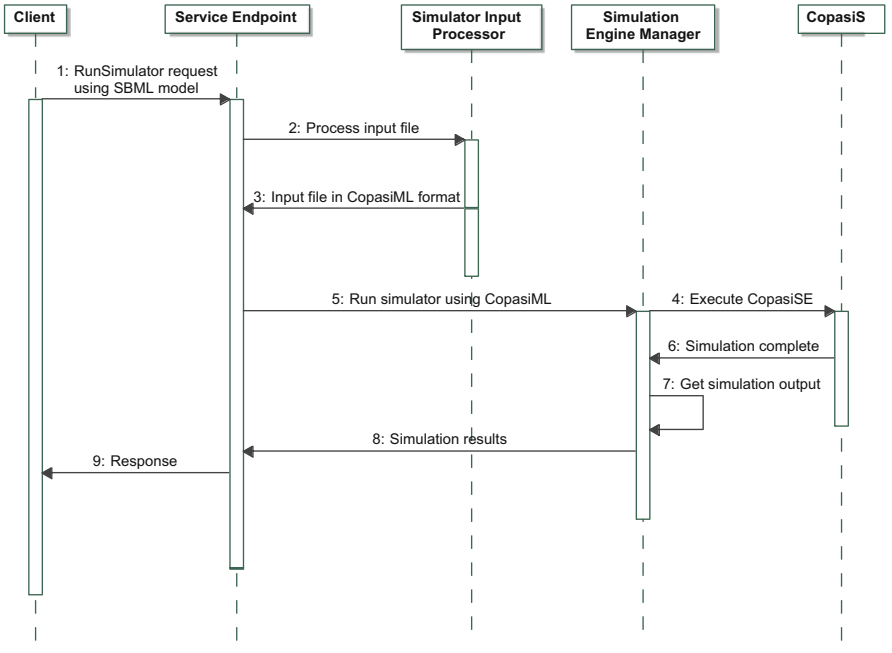

Fig. 4. Sequence of messages between components of layers in CopasiWS

toolkit [23] to generate the interface definition of the services in Java from their WSDL as discussed in section 2 and then added each service interface implementation code to complete the coding process. The services run in an Apache Axis 2 engine, which is hosted on an Apache Tomcat server [24]. All the components in the logic layer excluding the scheduler (the third party component) are also developed in Java.

The CopasiSE that resides in the resource layer is the command line version of COPASI (originally written in C++, though only the executable binary is used here) and runs on the actual server; alternatively it can run on a Condor pool or on a grid system. The user database is presently implemented using a file system. Future versions of CopasiWS will likely use a database management system for the user database.

Interested users should consult [25] to obtain the WSDL for simulation services presently available in CopasiWS. Currently there is no requirement for users to register before using the CopasiWS synchronous version. However those interested in using the asynchronous version should contact the authors to obtain username and password.

4.1 Web Portal User Interface (CopasiWeb)

To test the range of services available in CopasiWS, we developed a web portal user interface (CopasiWeb). CopasiWeb is one of the possible clients of

CopasiWS. It is basically divided into two parts: the first part is the simple HTML page that appears in user's browser and provides a means of interacting with the CopasiWS, while the second part is the Web service manager, which is hosted on a Tomcat Server. This converts the user's request from the browser into Web service calls that are directed to the CopasiWS. CopasiWeb is based on the Model View Controller (MVC) architecture and is implemented using Apache Struts 2 [26]. A detailed description of CopasWeb is outside the scope of this paper.

CopasiWeb is available from [27]. There is no requirement for users to register to use it, except if they want to execute the asynchronous services as described above.

4.2 Example Usage

CopasiWS provides a suite of services that can easily be combined with other applications and services to provide a flexible platform for modeling, simulation and analysis of biological processes. The steps involved in running CopasiWS depend on which version (synchronous or asynchronous) a client wants to use. For the synchronous version, the communication with the service follows a simple request and response approach. The client sends a run simulator request to the service and waits for a response. In the case of asynchronous version, client needs to follow a sequence of steps to execute the service. Here we use parameter estimation Web service to illustrate how to use the asynchronous version.

The parameter estimation task/service as earlier described is used to estimate the values of the unknown parameters in biochemical models. To carry out model parameter estimation process, the user needs to make the following available to the service:

1. a biochemical model in SBML format;
2. experimetal data in SBRML format;
3. information about the model parameters to estimate (i.e. model parameter identifier with lower and upper boundary values);
4. optimization method and its parameter values.

The above data are then passed to the service by the client application through the following sequence of steps using appropriate service operations:

1. Client creates a simulation resource using username and gets resource ID in return;
2. Client uses the resource ID to submit biochemical model in SBML;
3. Client submits experimental data in SBRML format using the resource ID;
4. Client sets the model parameter items to estimate and optimization method to use using the resource ID;
5. Client starts the simulator using the resource ID;
6. Client periodically checks the simulation status (polling approach) using the resource ID;
7. Client gets the simulation results (usually updated SBML model);
8. Client destroys the created resource. A resource that is not destroyed by the client will be destroyed by the system after it's lifetime.

This sequence of steps can easily be automated using a workflow management system. For instance a systems biology workflow can be constructed to retrieve the SBML model from the Biomodels database [7] and experimental data in SBRML from other Web services and then use these as input to the parameter estimation service to estimate the unknown SBML model parameters. Each of these tasks would be operating from a different server on the Internet.

5 Conclusion and Further Work

Systems biology is an increasingly popular mode of research in the biological sciences which makes heavy use of computational methods and resources. We have constructed a collection of Web services, named CopasiWS, that expose the functionalities of the systems biology modeling and simulation software COPASI. To our knowledge this is the first implementation of systems biology simulation tasks conforming to Web services standards. The availability of such methods through the means of Web services will allow a more flexible approach to computational systems biology where data and services are distributed throughout the Internet. An obvious use of these Web services would be to provide the simulation tasks that are currently only available through a graphical user interface in a client–server model where heavy computational resources are hosted behind this interface. Another new computing paradigm that these Web services allow, is the construction of distributed workflows, for example using Taverna [10], which is is already widely used in the related field of bioinformatics. Furthermore this will also open up the possibility of running complex simulations across the network. For example this interface could easily be exploited to construct a multi–scale simulation environment where the COPASI tasks fulfill one of the levels (e.g. cellular) while other Web services, or Web services clients, implement other levels (e.g. tissue or organ).

Future work will consist of implementing access security features and further user management functions. Because it is likely that computing power needs will increase, we plan on implementing methods to access Grid resources in the logic layer. Additionally, we would like to demonstrate the power of combining Web services by creating distributed workflows that access other resources, such as the Biomodels database.

Acknowledgments. We thank Stefan Hoops, Douglas Kell, Peter Li, Norman Paton, Irena Spasić and Neil Swainston for many helpful discussions. We greatly benefited from the lessons taught by Anwar Ul Haq's first prototype of COPASI Web services. We thank the generous financial support by the BBSRC and EPSRC to this project through funding of the Manchester Centre for Integrative Systems Biology. The MCISB is a Centre of the Biotechnology and Biological Sciences Research Council.

References

[1] Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., Kummer, U.: COPASI — a COmplex PAthway SImulator. Bioinformatics 22, 3067–3074 (2006)

[2] Funahashi, A., Tanimura, N., Morohashi, M., Kitano, H.: CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. Biosilico 1, 159–162 (2003)

[3] Sauro, H.M., Hucka, M., Finney, A., Wellock, C., Bolouri, H., Doyle, J., Kitano, H.: Next Generation Simulation Tools: The Systems Biology Workbench and BioSPICE Integration. A Journal of Integrative Biology 7(4), 355–372 (2003)

[4] Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., Kanehisa, M.: KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Research 27(1), 29–34 (1999)

[5] Baxevanis, A.D.: Searching NCBI databases using Entrez. Current protocols in bioinformatics 24, 1.3.1–1.3.26 (2008)

[6] Labarga, A., Valentin, F., Anderson, M., Lopez, R.: Web services at the European bioinformatics institute. Nucleic Acids Research 35, W6–W11 (2007)

[7] Le Novére, N., Bornstein, B., Broicher, A., Courtot, M., Donizell, M., Dharuri, H., Li, L., Sauro, H., Schilstra, M., Shapiro, B., Snoep, J., Hucka, M.: Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Research 34, D689–D691 (2006)

[8] Le Novére, N., Finney, A., Hucka, M., Bhalla, U.S., Campagne, F., Collado-Vides, J., Crampin, E.J., Halstead, M., Klipp, E., Mendes, P., Nielsen, P., Sauro, H., Shapiro, B., Snoep, J., Spence, H., Wanner, B.: Minimum Information Requested In the Annotation of Biochemical Models (MIRIAM). Nature Biotechnology 23(12), 1509–1515 (2005)

[9] Systems Biology Ontology (SBO), `http://www.ebi.ac.uk/sbo/`

[10] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research, W729–W732 (2006)

[11] Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana Workflow Environment: Architecture and Application. In: Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (eds.) Workflows for e-Science. Scientific Workflows for Grids, pp. 320–339. Springer, London (2007)

[12] Mendes, P.: GEPASI: A software package for modeling the dynamics, steady states and control of biochemical biology and other systems. Computer Application in the Biosciences 9(5), 563–571 (1993)

[13] Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., Cuellar, A.A., Dronov, S., Gilles, E.D., Ginkel, M., Gor, V., Goryanin, I., Hedley, W.J., Hodgman, T.C., Hofmeyr, J.H., Hunter, P.J., Juty, N.S., Kasberger, J.L., Kremling, A., Kummer, U., Le Novére, N., Loew, L.M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E.D., Nakayama, Y., Nelson, M.R., Nielsen, P.F., Sakurada, T., Schaff, J.C., Shapiro, B.E., Shimizu, T.S., Spence, H.D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., Wang, J.: The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. Bioinformatics 19, 524–531 (2003)

[14] Web Services Interoperability Basic Profile, 1.1, http://www.ws-i.org/Profiles/BasicProfile-1.1.html
[15] Bornstein, B.J., Keating, S.M., Jouraku, A., Hucka, M.: LibSBML: An API Library for SBML. Bioinformatics 24(6), 880–881 (2008)
[16] Dada, J.O., Paton, N.W., Mendes, P.: Systems Biology Results Markup Language — Structure and Facilities for Systems Biology Results Representation (SBRML Specification) (2008), http://www.comp-sys-bio.org/tiki-index.php?page=SBRML
[17] Thain, D., Tannenbaum, T., Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience 17, 2–4 (2005)
[18] Petzold, L.: Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. SIAM Journal on Scientific and Statistical Computing 4(1), 136–148 (1983)
[19] Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
[20] Fell, D.: Understanding the Control of Metabolism. Portland Press, London (1996)
[21] Mendes, P., Kell, D.: Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. Bioinformatics 14(10), 869–883 (1998)
[22] Java programming language, http://java.sun.com
[23] Axis toolkit site, http://ws.apache.org/axis/
[24] Tomcat Apache servlet site, http://tomcat.apache.org/
[25] Copasi Web Services Services, http://turing.mib.man.ac.uk:8080/CopasiWS/services/listServices
[26] Apache Struts 2, http://struts.apache.org/2.x/
[27] CopasiWeb: Web Portal Interface to CopasiWS, http://turing.mib.man.ac.uk:8080/CopasiWeb/CopasiWebUI/
[28] Li, P., Oinn, T., Soiland, S., Kell, D.B.: Automated manipulation of systems biology models using libSBML within Taverna workflows. Bioinformatics 24(2), 287–289 (2008)