A Specification Language and a Framework for the Execution of Composite Models in Systems Biology

Ofer Margoninski, Peter Saffrey, James Hetherington, Anthony Finkelstein and Anne Warner*

Centre for Mathematics and Physics in the Life Sciences and Experimental Biology (CoMPLEX) , University College London, Gower Street, London, WC1E 6BT
omargoni@cs.ucl.ac.uk

Abstract. When modelling complex biological systems it is often desirable to combine a number of distinct sub-models to form a larger composite model. We describe an XML based language that can be used to specify composite models and a lightweight computational framework that executes these models. The language supports specification of structure and implementation details for composite models, along with the interfaces provided by each sub-model. The framework executes each sub-model in its native environment, allowing extensive reuse of existing models. It uses mathematical and computational connectors and translators to unify the models computationally. Unlike other suggested approaches for model integration, our approach does not impose one modeling scheme, composition algorithm or underlying middleware framework. We demonstrate our approach by constructing a composite model describing part of the glucose homeostasis system.

1 Introduction

Recent years have seen the proliferation of mathematical models used to describe biological phenomena. Among others, models have been proposed for describing metabolic processes, signalling pathways, transport processes and various electro-physiological systems. While many detailed models describing various biological aspects have been suggested, very few models describe a complete physiological system, organism or organ, across scales. Such large scale models can, theoretically, be created by integrating together existing detailed models describing sub-aspects of the desired system[12], but the lack of suitable tools for model integration in Systems Biology has made this task, so far, nearly impossible. This paper describes such a tool. In presenting this tool, we use the structured view of models and modelling activity of the *meta-model* suggested by Finkelstein et. al. in [1].

Models exist in a great variety of schemes and formats including differential equations, stochastic and process algebra models, each of which have their attendant facilities and tool support. However, it is possible to view each model

* We thank the DTI for supporting this research through the UCL Beacon Project

as a specification for possible computations. Each model can be executed by a software tool, or an *engine*. Using the model and a set of inputs, or *context* the engine provides a set of outputs, or an *interpretation*. By mapping the outputs of one model into the inputs of another, we can construct a *composite model*. Repeatedly composing such models together can produce *composite models* that are arbitrarily large and complex. The basic building blocks of such models, which can not be further decomposed are known as *elementary models*. Elementary models are constructed in a modelling environment such as Mathematica or XPPAUT.

1.1 The Composite Model Description Language

We have developed an XML based language, the Composite Model Description Language (CMDL), that allows the description of composite and elementary models, so that they can be used within our framework. CMDL also enables the specification of composite models themselves.

For all models, A CMDL file can be used to describe the functionality and the interfaces provided by the model. CMDL allows each sub-model to have multiple interfaces to capture models that have more than one functionality, for example: an ODE model can be solved to plot its dynamical variables versus time, or analyzed to find its bifurcation points. CMDL also can be used to provide attribution annotation and to link the behaviour of a model to the biological phenomena, or *aspects* it represents, to allow for more convenient collation and reuse. Thus, CMDL is MIRIAM[25] compliant. As suggested in the MIRIAM proposal, we use existing ontologies to minimise ambiguity.

For composite models, A CMDL description also specifies the model architecture and implementation details: What sub-models are used, how they are connected together, and in what order they should be executed.

CMDL has been designed to provide biological, mathematical and computational information about a model and to make that information easily accessible to all parties.

1.2 The Computational Framework

We have also developed a lightweight computational framework that enables the execution of composite models specified in CMDL. Individual models are executed on their native tools and are integrated by the framework. Usually they need not be modified in order to be used by the framework. The framework utalizes *translators* to take account of inevitable differences in input and output formats, including differences in timescale. It uses *smart connectors* to resolve any feedback present in the composite model structure. Each part of the framework is generic and based on well defined interfaces, so it can easily be replaced by user-defined algorithms and translations. Our framework utilizes existing middleware infrastructure, such as dynamic link libraries or Web Services[37] for communicating between the different components. We do not

presume the existence of any specific infrastructure, and the framework can potentially run on many different middleware infrastructures. At the moment we support only the integration of ODE models, but the framework can be easily extended to support the integration of models developed in other schemes by the development of appropriate connectors.

This paper proceeds as follows. Firstly, we review the current state of the art in model integration in Systems Biology. We then describe the Composite Model Description Language and the model integration framework in more detail. We conclude by describing a composite model of glucose homeostasis that we have specified and executed using the framework.

2 Related Work

2.1 Approaches originating in Systems Biology

At the moment, there exists no component middleware specifically designed for the integration of models in Systems Biology. While our proposal shares some concepts with Cell-ML[9],[23], SBML[28],[29] and the Systems Biology Workbench (SBW)[31], it also differs from them in several crucial aspects.

CellML[9],[23] was designed with the view of enabling modellers in Systems Biology to specify composite models composed of a number of sub-models. CellML requires the definition of input and output variables for each model; model composition is achieved by connecting inputs and outputs from separate models. CellML, however, does not allow specification of how the composed model should be executed, detailed descriptions of the model interfaces, or the integration of models which are not ODE models, or which are specified in a format other then CellML. Thus, while CellML may be quite adequate for the conceptual representation of a composite model, composed of several ODE based sub-models, it does not, currently, fulfil our need for a composite model description language, which is more generic, on the one hand, and implementation oriented, on the other hand.

The other currently prominent modelling language for Systems Biology is SBML - The Systems Biology Markup Language[28],[29]. It attempts to standardize the expression of ODE based models of cellular systems, concentrating on chemical reactions. SBML is a rich language in this environment and has good take up within the community. However SBML currently does not allow for modularization, has no support for interface specification, and does not support linkage with models created in other tools and languages. Thus, SBML can not be used to integrate existing, heterogeneous models. We view our proposal as complementary to SBML. We use the SBML annotation scheme, and existing SBML models can be easily wrapped with a CMDL description in order to facilitate their integration with other models.

SBW[31] is a generic middleware for the integration of software tools, used in Systems Biology. It was not designed specifically to facilitate the integration of models, and is actually a quite generic middleware architecture, similar to CORBA[10].

Several tools exist to support model construction and simulation specifically in Systems Biology. E-Cell[17],[34],[33] is a whole-cell and multi-cell simulation tool based on an object oriented approach . While it enables the creation of models using a few different schemes, such as reaction-diffusion, S-System and flux distribution analysis schemes, it does not support the integration of models created in other tools. Also, currently only one connecting algorithm, which is embedded in the software itself, is supported.

The XS-system[2] enables the construction of models of cellular networks from a set of building blocks representing syntheses, degradations, reversible reactions and enzymatic reactions. The resulting model is represented as a set of ODEs, specifying the rate equations for the various substances involved. Representation using SDEs (stochastic Differential Equations), timed automata or hybrid automata is also supported, but in a rather limited manner. The XS-system is designed to support the construction of models from a pre-existing set of existing, elementary, building blocks rather than allowing the user to integrate models created in different tools.

BioSpice[5], [6] is a collaborative project of American universities and research centres. It aims to build a comprehensive software environment that integrates a suite of analytical, simulation and visualisation tools related to cellular systems biology. At the moment, the tool suite focuses on individual model construction and analysis and does not address model integration.

2.2 Approaches originating in Software Engineering

There are a number of frameworks aimed at integrating heterogeneous components for simulation. The High Level Architecture (HLA) [21] is a general purpose architecture for simulation reuse and interoperability, developed for the Defense Modeling and Simulation Office (DMSO). HLA uses a central service to coordinate a number of models via a standard time-step interface. However, there is no explicit language to describe model connections and only the time-step interface is supported.

Generic component frameworks, such as CORBA[10], COM[11], Java Beans[18], and more recently, Web Services[37] include an Interface Definition Language (IDL), such as IDL for CORBA and COM, and WSDL[38] for Web Services, used to specify the functional interfaces exposed by the components. Process execution languages, such as BPEL-WS[4] , enable a multiple component execution to be specified. These frameworks do not, however, allow detailed annotation of the nature of each component necessary both for heterogeneous model integration and understanding of biological models. They also have poor support for specifying the architecture of the overall model.

The concept of interconnecting components exist in many Architectural Description Languages (ADLs), for example Darwin[13], [14], [15] and Wright[39], including the possibility of a rich set of component connectors. However, these languages can not be used for specifying the biological aspects the models represent, or how the models should be executed.

The Unified Modelling Language [35],[36] is a very comprehensive model description language, designed mainly for modelling software systems. The main focus is on modelling in detail the code itself and not the aspects, or phenomenon, that the code relates to. It is also difficult, in UML, to present an overall view of the different functionalities a certain model has, as opposed to a detailed representation of its interfaces. UML is also not well suited for representing the overall component architecture of a system — ADLs are better suited for this purpose.

We focus on the integration of currently existing approaches and techniques in Systems Biology, such as ontologies and model description languages, with Software Engineering tools and techniques such as ADLs, IDLs, process execution languages and component frameworks. Through this integration we build a framework for the representation and execution of composite models in Systems Biology.

2.3 Approaches originating in other scientific fields

The General Coupling Framework, GCF[19] enables the creation of composite models out of individual model components, developed in a variety of programming languages. Like CMDL, GCF supports the description of the interfaces of the individual components, as well as the architecture of the overall model. Unlike CMDL, GCF focuses on the integration of software modules written in programming languages such as C, Fortran or Java, and requires 'put' and 'get' calls to be placed into the individual modules source code before they can be used within the framework. Currently GCF uses a time-stepping algorithm, embedded within the architecture, in order to perform the simulation.

The Cape-Open standard[8] is a specification for a collection of middleware interfaces, aimed at enabling the integration of models and modelling tools in the chemical industry. The interfaces enable the integration of different Unit Operations Modules, modelling the activity of a unit operation within a chemical plant, and numerical solvers, within the same simulation environment, called the Simulator Executive. There is no proposed standard for the specification of composite models, and it is assumed each Simulator Executive would use its own proprietary methods for that.

3 The Composite Model Description Language

The Composite Model Description Language (CMDL) is an XML schema for *model description files*. For all models, the CMDL file contains a section describing the biological phenomenon described by the model, a section describing the functionalities and interfaces provided by the model, and a section describing some relevant meta-data. For composite models, the model description file also includes a specification of how the model should be implemented: What submodels are included, how they should be connected together, what connectors should be used, and in what order should they be executed.

3.1 The Model Interface Description

The top level element in any CMDL file is the model element, which has an id attribute. A model contains meta-data, phenomenon and functionality elements as it's immediate sub-nodes. A composite model would also contain sub-models elements, which are used to specify the submodels used in the model.

Meta-data The Meta-Data section contains the attribution annotation, as required by the MIRIAM[25] standard proposal. It includes a citation of the reference description or scientific paper with which the model is associated, details of the model creators, date and time of creation and a statement about the terms of distribution. These details are specified using RDF, in the same manner as in SBML[9],[23] models.

phenomenon The phenomena element links the model with the biology it represents. It is used to precisely specify what biological phenomena are described by the model, in accordance with the MIRIAM[25] proposal. It is composed of a list of phenomenon elements describing the biological processes depicted by the model. Each phenomenon element contains a textual description, and possibly one or more references to terms from the same or different ontologies, which together serve to define the phenomenon. We use the SBML annotation element[30] to refer to these terms. For example, a model describing Insulin stimulus of hepatocytes and the resulting signalling cascade will include the phenomena "Detection of hormone stimulus" (GO term 9720), and "Insulin receptor signalling pathway" (GO term 8286).

A phenomenon element also contains compartment and aspect elements. They are used to specify where within the organism the mentioned phenomenon, or process, occurs and the concrete *measurables* that the model describes. These measurables are the main modelling results, to be compared to the results obtained in experiments or by executing other models for validation purposes. They usually correspond to the main variables imported and exported by the model. Aspects specified by the sub-models, which may be of less interest for the overall model, need not be listed.

While our concept of compartments is similar to that of SBML, and the SBML species element can be viewed as a subtype of our aspect element, we use compartment and aspect elements only for annotation and not for the actual specification of the model itself. Compartments and aspects may again be specified by making references to terms in various relevant ontologies. For example, the cytoplasm of a hepatocyte can be specified by the combination of terms hepatocyte(CELL:OBO term 182), and cytoplasm (GO term 5737) .

functionalities The functionalities section serves to describe the interfaces provided by the model. It describes, given what inputs, what outputs are provided by the model. The same model may be interpreted in many different ways, using the same or different sets of inputs and engines, to give different predictions

or results. For example, an ODE model can be run in a simulation, when provided with all the required parameter values, or analyzed for its null clines and bifurcation points. Thus, a model may provide several different *functionalities*. These functionalities may be specified at different levels of detail: At the highest level, the *broad functionality* level specifies what predictions a model can make (output aspects) based on what data (input aspects). This level of specification provides a basic summary of model behaviour that may be useful to biologists in particular.

The next level is the *mathematical functionality* level, which is used mainly to specify the mathematical format of the model's inputs and outputs. This is done by assigning at least one *variable* to each aspect. A variable in CMDL is used to provide, usually quantitative, information regarding an aspect. For example, a variable may serve to describe the concentration of Calcium over time or the frequency of Calcium oscillations. A variable can be of many different types - It can be, for example, a scalar, a vector, tensor, a matrix, a probability distribution or a time track - describing how the value of a dynamical variable is changing with time. By assigning a variable of a certain type to an aspect, we specify exactly how this aspect is described, mathematically, by the model. MathML may be used to describe the precise format of a variable. For example, We can use MathML to specify that $Ca = F(t)$, where $t0 < t < t1$. A variable should have units, unless the dimension it describes is 'dimensionless' such as, 'the number of particles' .

The mathematical functionality also includes a list of required parameters. The difference between parameters and variables is that usually the value of parameters remains fixed during the course of an interpretation of a model, while the values of variables may change. Currently we use the SBML syntax for specifying parameters. However, the parameter value may be specified in a separate, auxiliary, parameter values file. The enables the framework to run multiple instances, or copies, of the same model with different parameters.

A mathematical functionality may also specify the mathematical scheme in which the model is implemented: For example, chemical reactions can be mathematically described either deterministically as a set of ODE's or stochastically using Gillepsi's algorithm.

The mathematical functionality description level is useful both to biologists and mathematicians using the model.

The most detailed level is the *computational interface* description. The computational interface specifies the type of interface supported - for example Web Services, DLL libraries or a simple output file, the name of the interface supported, and a reference to a file containing the actual interface specification. This would be a WSDL file for Web Services, or a C/C++ header file in the case of a DLL library. These files specify the precise data structures of the variables involved. In the case of Web Services, the WSDL file also specifies the location of the sub models to be used by using the WSDL 'binding' element.

The interface referenced should be one which is currently supported by the orchestrator and at least some existing connectors, in order for the model to be

used within our framework. Information at this level is for use by the computer scientists responsible for implementing a composite model using this model as one of its components.

A broad functionality may contain several different mathematical functionalities and each mathematical functionality may in turn contain several different computational interfaces. Thus, the different functionalities for each model form a tree hierarchy.

3.2 Specifying Composite Models

In order to execute a composite model we need to specify what model instances, translators and connectors should be used. We then need to map out the connections between these elements, specifying how the inputs required by each model instance are satisfied. Finally, we need to specify in what order the model instances and the connections should be invoked, and which specific computational interfaces should be called. Thus, in a manner similar to BPEL[4] we provide a process description notation with many features reminiscent of an executable language. Like an executable programming language a CMDL model is unambiguous, provided that all of the internal models it is composed of are unambiguous. In other words, a CMDL model will always yield the same results for a specific set of inputs, provided that the elementary models it is composed of behave in this manner. The key difference between languages such as CMDL or BPEL and programming languages used to describe executable internal processes is that a CMDL or BPEL file also calls for the execution of internal processes, or in our case elementary models, without specifying how these internal processes or elementary models actually handle the data - this is assumed to be specified by the modelling language in which the elementary model is specified.

While the CMDL specification is detailed enough to support execution of the composite model by our framework, it is also designed to enable mathematicians and biologists to gain a broad understanding of how the model is put together.

Specifying the Model Architecture The first thing to be specified is the model components to be used. A model component is an *instance* of a submodel executing on an engine, similar to the instance of an object in object oriented programming. Many instances can be created from the same submodel, perhaps using different parameters for each, each forming its own component. For example, in order to model a liver cell plate, comprised of many hepatocyte cells, one model component can be used to model each cell. All of these model components can be created from the same hepatocyte cell model, using the same or different sets of parameters. Different parameters may be used, for example, to reflect biological differences between periportal and periveneous cells.

Connections specify the topology of the network of models. *Horizontal connections* specify the connections between the sub-models - which output variables of which models are used as inputs for other models. *Vertical connections* map the variables of the overall model into the variables of the different components

it is composed of. For each variable mapping, we may specify a *translation*. This can be a simple scaling of the variable, or a more complex transformation. For horizontal connections which form a feedback loop, we also specify which smart connector should be used to resolve the feedback.

3.3 Specifying the algorithm used to solve the model

The algorithm for executing a composite model is specifed using a 'sequence' construct, similar to that found in BPELWS[4]. The sequence element contains a list of invocation elements. Each invocation element specifies either the invocation of a specific mathematical functionality on one of the pre-declared model components, or the invocation of a smart connector, used to solve several model components which are interdependent on one another. Currently, the only flow of control supported is a simple linear one. In the future we plan to support additional flow control elements already supported by BPELWS, such as those used to implement loops and branches.

4 Example of a Model Specification File

The appendix contains an example composite model specification file. The model described is of the generation of calcium oscillations in liver hepatocytes as a result of hormonal stimulation. The model depicted forms part of a more comprehensive model of this process, which will be described later.

The model file first defines the phenomena depicted by the model - the glucagon stimulated signalling cascade. The phenomena is defined both through a textual definition and through references to terms in the relevant ontologies. We also specify the compartments in which the phenomena of interest occurs - the hepatocyte membrane and cytoplasm. The last part of the phenomena element lists the actual aspects, or measurables, depicted by the model. In this case these are the activation level of G-Protein and the concentration of intracellular calcium.

The model is a composite model composed of two sub-models, listed within the 'submodels' tag. The first model describes the hormone binding to the G-Protein receptor, resulting in the release of PhosphoLipase into the cell, and the second model describes how PhosphoLipase causes Calcium oscillations.

The model has one functionality, predicting G-Protein activation levels and cytoplasmic concentration of Calcium, as a function of the concentration of Glucagon in the blood, over time. This functionality is specified in precise mathematical terms in the 'mathematical functionality' section: It provides timetracks of Calcium concentration and G-Protein activation level, and requires a timetrack of Glucagon levels. The units of the variables involved are also specified.

As we can see in the 'implementation' section, to implement this functionality, we create one instance of each sub-model, and then link together the two instances, feeding the PhosphoLipase concentration from the G-protein receptor model into the Calcium model, and feeding back the Calcium concentrations

from the Calcium model into the G-protein model. The model instances are linked together using a waveform relaxation connector. The sequence of execution steps for this functionality contains only one step - the execution of the connector.

5 The Model Integration Framework

Models specified in the CMDL language are executed by the orchestrator. The orchestrator serves mainly as a workflow co-ordination service. It executes the composite model by launching and executing the elementary sub-models on their respective engines, such as Xppaut and Mathematica, and passing data between them as required. The orchestrator communicates with the various engines through *engine wrappers* - pieces of software that expose the functionality of the different engines in a standard manner. The orchestrator uses *connectors* to solve together models which are interdependent on one another, and *translators* to carry out necessary data transformations between the models. The orchestrator is used in conjunction with a set of supporting information services, used to store data required for model runs, such as parameter values, as well as results obtained from model execution and, in the future, the CMDL files themselves. A separate paper about these information services is in preparation.

5.1 The core computational elements

The Engine Wrappers Individual instances of elementary models are executed by the software tools, or *engines*, in which they were originally developed. Accommodation of specific modelling tools within our framework is done through wrappers. Wrappers expose the functionality of the modelling tool in a standard way to the rest of the framework. Wrappers expose interfaces used to launch new model components and enable access to and the execution of computational interfaces of components already launched. Internally, the wrappers use the proprietary command set of the modelling tool in question to provide these operations. We currently have available wrappers for Mathematica[24], Xppaut [40], and for a C++ library used to numerically integrate differential equations using the numerical recipes[27] library.

The Orchestrator At the core of the framework is an *orchestrator*, which is used for executing composite models. The orchestrator serves mainly as a coordinator or process execution service. It reads the details of the composite model from the composite model specification file and then launches the sub-models and executes them according to the instructions provided in the file. The orchestrator maintains the global (composite) model variables, and passes them as inputs to the sub-models as required.

The composite model file may also specify the use of connectors and translators. These are called by the orchestrator in order to link together models, where the outputs of one model can not be linked to the inputs of the other model in a straightforward manner.

Connectors A connector serves to numerically integrate two models where such an integration is not trivial. For example, integrating two or more ODE models which are interdependent and which were implemented on different tools, as in our case studies. Several different connectors can be used to achieve the same task. For example, for integrating ODE models, one can use either a connector implementing a wave form relaxation algorithm[22] or a step wise integrator as described by [34].

The wave form relaxation algorithm uses a 'seed' function to guess the solution to one model and then iterates between solving each model, refining the overall solution to convergence. A step wise integrator runs all the models at once, performing the numerical integration using a method such as Euler's method or RangKutta. Each of these algorithms requires a different mathematical interface (see section 3.1).

The different connectors are best suited for use in different scenarios: A waveform connector typically executes at most a few dozen calls on each model, but each call is computationally intensive, as a complete simulation is performed, and requires the transfer of substantial amounts of data. A stepwise integration algorithm may make millions of calls on each model, but each call is computationally quick and requires the transfer of only a few values. Given these characteristics, a waveform connector may be more suitable for connecting together models residing on different, remote machines, while a stepwise integrator may prove to be more efficient when all models reside on the same machine.

Connectors may also be used in order to integrate together models developed in different schemes. For example, a connector can be built in order to connect together a Discrete Event (DEVS) model with an ODE model. Such a connector may generate events for the DEVS model when certain variables in the ODE model cross certain thresholds. It may also modify the values of certain parameters in the ODE equations when certain events occur. Such a connector may be used to link a DEVS model of an intracellular signalling system with an ODE model depicting gap-junctions and the flow of different chemical species through it. Similarly, connectors can be devised for stochastic models, based on suitable mathematical algorithms.

In addition to connectors, *translators* are used to take account of inevitable differences in input and output formats, such as differences in the units used by different models for the same variable, differences in timescale, or differences in the data structures used by the different model implementations.

The aim of this free-form approach is to allow each model component to be based on the most natural and appropriate scheme, rather than forcing each model into a unified system such as ODEs or discrete events.

To conclude, our framework is modular not only in the deconstruction of models, but in the components of the framework itself. Users of the framework are free to select from an existing range of connectors and translators or build their own, in order to achieve greater efficiency or the ability to integrate new types of models.

Underlying infrastructure Our computational framework does not presume any specific underlying middleware infrastructure. Currently, the engine wrappers we have built expose their interfaces either through dll library calls, or through web services[37]. Thus an orchestrator, or a connector, can both communicate efficiently with modelling tools residing on the local machine, and with modelling tools residing on other machines, perhaps in remote locations. Future engine wrappers may expose their functionality through other component middleware infrastructures, such as COM, CORBA[10] or SBW[31] .

5.2 supporting services

The computational framework is supported by a number of *information services* that provide information about each model used during the integration and then collect the results during a model run. Parameters required for the interpretation of a model are obtained from the context service, which serves as a central repository for parameter values to be used in biological modeling. The results, or *interpretations*, of the models are stored by the interpretation service. We envision a central model repository, such as 'BioModels.net'[7] being used to store existing models. One should be able to systematically search the repository in order to find desired sub-models required for the creation of a new composite model. We have implemented prototype versions of the context and interpretation services. Their functionality is exposed both via web services, to support communication with the rest of the framework, and through a web based user interface, which enables users to manually query the services for parameter values, or the results of previous model runs. The context and interpretation services will be described in greater detail in a future publication.

5.3 Example

Figure 1 shows a view of our *model integration framework*, which is used to execute *composite models*, specified in CMDL. In the figure we can see the two sub-models of the model depicted in section 4, executing on their respective tools, Xppaut and Mathemtatica. Since the two models are interdependent, they are integrated by means of a connector. The orchestrator is responsible for launching the models and the connector, for passing to them the necessary input values and model parameters, and for collecting the results and storing them on the interpretation service.

6 Using CMDL to model glucose homeostasis

We are part of a research project at University College London whose aim is to produce a physiological model of the liver which is integrated across scales[41]. As part of the project, we have recently used CMDL, along with our model execution framework, in order to specify and execute a composite model describing glucose homeostasis[42]. Glucose is the readily available fuel, or source of energy, which is

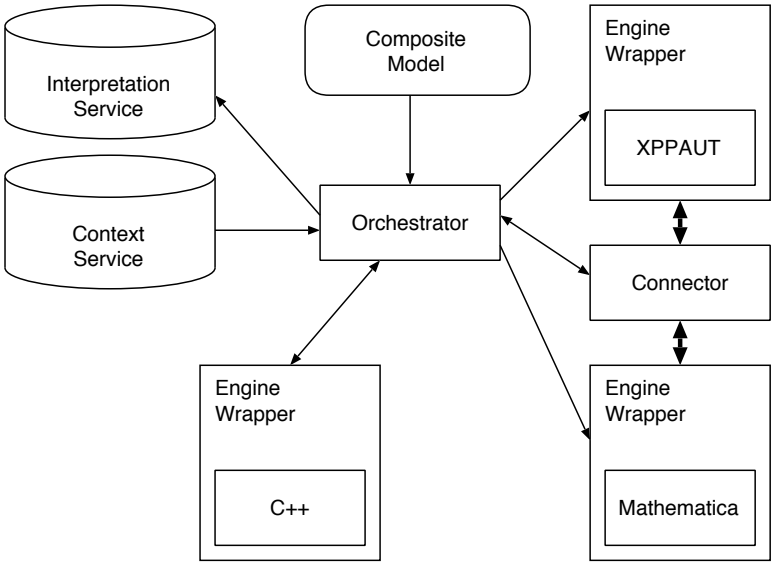

Fig. 1. Model Integration Framework

being supplied by the blood to all cells in the body. Glucose is stored in the liver, in the form of glycogen. Glycogen buildup and release in the liver is controlled by two hormones, Insulin and Glucagon, secreted by the Pancreas.

Our model is able to predict glucose levels in the blood, as a function of the dietary regime, and various other parameters, such as the affinity of liver cells receptors to Insulin. We have used new and existing models to create our composite model of glucose homeostasis. Currently our model includes basic models of hormone secretion by the Pancreas and glucose transport in the blood stream, along with a quite detailed model of glucose release or intake by the liver, as a function of current glucose and hormone levels in the blood. This detailed model is in turn composed of 5 sub-models, describing the membrane receptors, the second messengers responses within the cell, and the actual build-up and breakdown of glycogen. The model and the sub-models it is composed of will be described in detail in a subsequent publication.

As these models are interdependent on each other and form a feedback loop, we have used a waveform relaxation connector in order to solve them together. The models also use different units and different time scales. We have thus used two simple scaling translators when connecting the models. One is used for adjusting the time scales between the different models, and the other is used to scale the other, time dependent variables.

By replacing some of the model components with simpler or more elaborate models, we try to determine how sensitive is glucose homeostasis to details in the description of the different sub-systems involved, specifically the receptor

mechanisms and second messenger cascades in liver hepatocytes. We are trying to ascertain whether a good approximation of glucose homeostasis can be achieved by using relatively simple models of those sub-systems, or whether a detailed, mechanistic model of them is required. By removing or changing some of the model components, we try to ascertain what is the role of the different control mechanisms, such as hormonal control and direct glucose control, in maintaining glucose homeostasis.

7 Conclusions

We have presented CMDL, an XML Composite Model Description Language. CMDL integrates elements from ontologies, mathematical description languages such as MathML, interface description languages such as WSDL[38], Architecture Description languages and process execution languages, in order to provide a full description and specification of composite models in Systems Biology, moving from a broad description of the phenomenon depicted by the model and the functionalities provided by it, through an architectural description of the implementation, down to the precise details required for model execution. While we have borrowed heavily from well known techniques used to describe these different levels, we are currently unaware of any other attempt to integrate them together in a similar manner.

We have also presented a lightweight computational framework that is able to execute composite models specified in CMDL. It enables the integration of models, executing on a variety of different tools, and potentially executing on different machines in different locations. Unlike some other currently existing frameworks, our framework does not assume a specific model integration algorithm. Different connectors and translators can be used to connect the models together, and model composers can select the connector or translator which is most suitable to the task at hand and to the available facilities and model executing tools. The same overall composite model architecture can be implemented in radically different ways - the sub-models can be integrated using a C++ framework, with all models running on the same machine, or be integrated using Web Services, with different models running on distributed machines.

We have used our framework to implement a composite model of glucose homeostasis, composed of several existing models. These models run on a variety of different time scales, use different units for the variables involved, and are interdependent on one another. To tackle those issues, we have used a variety of connectors and translators. This model is now being actively used for scientific exploration.

One of our principal aims was to build a model integration framework which is easy to use. Unlike other suggested frameworks, such as SBW[31] or GCF[19], our framework does not require the writing of any program code on behalf of the modellers. We are currently building graphical user interface tools that can be used to specify and display the composite model specification files. Even without these tools, only about two days of work were required to write the CMDL files

for the composite glucose homeostasis model, and launch the integrated model, by a person who was not familiar before-hand with XML or CMDL.

While up until now we have used our framework mainly to integrate ODE models, with the provision of suitable connectors and translators, our framework can be used to integrate discrete event, and perhaps process algebra models, as well.

References

1. A. Finkelstein, J. Hetherington, L. Li, O. Margoninski, P. Saffrey, R. Seymour, and A. Warner "Computational Challenges of Systems Biology", in *IEEE Computer* 2004, vol. 37(5), pp. 26-33
2. M. Antoniotti, F. Park, A. Policriti "Model building and model checking for biochemical processes", in *Cell Biochemistry and Biophysics* 38, 2003 pp 271-186
3. E. Bugianesi et. al. "Quantification of Gluconeogenesis in Cirrhosis: Response to Glucagon" GASTROENTEROLOGY 1998; Vol. 115 pp. 1530-1540
4. Specification: Business PRocess Execution Lanugage for Web Services Version 1.1 Available at: `http://www-106.ibm.com/developerworks/library/ws-bpel/`
5. BioSpice, at `https://community.biospice.org/`
6. B. Mishra et. al. "A Sense of Life: Computational and Experimental Investigations with Models of BioChemical and Evolutionary processes" in *OMICS* Vol. 7(3) 2003
7. The BioModels database at
   t http://www.ebi.ac.uk/biomodels/
8. The CAPE-OPEN Laboratories Network, at `http://www.colan.org/`
9. CellML at `http://www.cellml.org`
10. The OMG Corba's web site - `http://www.corba.org`
11. The Distributed Component Object Model, by Microsoft, at `http://www.microsoft.com/com/tech/DCOM.asp`
12. P. V. Coveney and P. W. Fowler "Modelling biological complexity: a physical scientist's perspective" in *Interface* 2005, vol. 2, pp 267-280
13. J. Magee, N. Dulay and J. Kramer "Structuring Parallel and Distributed Programs", in *IEE Software Engineering Journal* Vol 8, No. 2 March 1993 pp 73-82
14. J. Magee, N. Dulay, S. Eisenbachand J. Kramer "Specifying Distribute Software Architectures" in *Proceedings of the Fifth European Software Engineering Conference, ESEC 95* 1995, pp 137-153
15. J. Kramer and J. Magee "Exposing the Skeleton in the Coordination Closet", in *Proceedings of the Second International Conference on Coordination Languages and Models* 1997 LNCS vol. 1282 pp 18-31
16. Dublin Core Metadata Initiative, at `http://dublincore.org/`
17. `http://www.e-cell.org`
18. Enterprise Javabeans Technology `http://java.sun.com/products/ejb/`
19. R.W. Ford et. al. "GCF: A General Coupling Framework" in Concurrency and Computation: Practice and Experience, 2006 vol. 18 pp. 163-181
20. Gene Ontology Consortium "An Introduction to Gene Ontology" at `http://www.geneontology.org/GO.doc.html`
21. The High Level Architecture `https://www.dmso.mil/public/transition/hla/`
22. Linzhong Li and Steve Baigent, "Integrating biosystems using waveform relaxation", submitted to bioinformatics

23. Catherine M. Lloyd, Matt D. B. Halstead and Poul F. Nielsen, CellML: its future, present and past, Progress in Biophysics and Molecular Biology, Volume 85, Issues 2-3, June-July 2004, Pages 433-450. (http://www.sciencedirect.com/science/article/B6TBN-4BT1658-2/2/109054184e74743e7ad3371bae71dd56)
24. Mathematica, Wolfram Research, described at `http://www.wolfram.com/products/mathematica/index.html`
25. N. L. Novere, A. Finney et. al. "Minimum information requested in the annotation of biochemical models (MIRIAM)", 2005, Nature Biotechnology, vol 23 pp. 1509 - 1515.
26. O. L. Munk et. al. "Liver Kinetics of Glucose Analogs Measured in Pigs by PET: Importance of Dual-Input Blood Sampling", Journal of Nuclear Medicine, 2001, vol 42 pp. 795-801
27. W. H. Press et. al. "Numerical Recipes in C: The Art of Scientific Computing", Cambridge University Press, 1992
28. SBML: Systems Biology Markup Language, 2003; at `http://www.sbml.org`
29. , M. Hucka et. al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models" Bioinformatics Vol. 19 no. 4 2003 PP 524-531
30. N. L. Novere, A. Finney "A simple scheme for annotating SBML with references to controlled vocabularies and database entries", 2005, available at `www.ebi.ac.uk/compneur-srv/sbml/proposals/AnnotationURI.pdf`
31. SBW: The Systems Biology Workbench Project, at `http://www.sbw-sbml.org/the_project.html`
32. S. Schuster, M. Marhl, T. Hofer "Modelling of simple and complex calcium oscillations" em European Journal of Biochemistry 2002, vol 269(5) page 1333
33. K. Takahashi et. al. "Computational Challenges in Cell Simulation: A Software Engineering Approach", in *IEEE Intelligent Systems*, pp 64 - 71, SepetemberOctober 2002
34. "A multi-algorithm, multi-timescale method for cell simulation", K. Takahashi, K. Kaizu, B. Hu, M. Tomita, *Bioinformatics*, 20(4), 538-546, (2004).
35. G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1999
36. The Unified Modeling Language, a specification of the Object Management Group `http://www.uml.org`
37. "New to Web Services", by IBM, at `http://www-106.ibm.com/developerworks/webservices/newto/websvc.html` The W3C Web Services Activity at `http://www.w3.org/2002/ws/`
38. Web Services Description Language at `http://www.w3.org/TR/wsdl`
39. R. Allen, D. Garlan "A Formal Basis for Architectural Connection" in *ACM Transactions on Software Engineering and Methodology* July 1997
40. Bard Ermentrout, "Simulating, Analyzing, and Animating Dynamical Systems: A Guide to Xppaut for Researchers and Students", SIAM, 2002
41. The UCL Beacon Project, 2003; http://grid.ucl.ac.uk/biobeacon/php/index.php
42. Peter J. Klover and Robert A. Mooney, "Hepatocytes: critical for glucose homeostasis", *The International Journal of Biochemistry and Cell Biology*, Volume 36, Issue 5, May 2004, Pages 753-758.

8 Appendix - the Hepatocyte Glucagon G-Protein Calcium model encoded in CMDL

```
<?xml version="1.0" encoding="UTF-8"?>

<model id="Glucagon_GProtein_Calcium"
    xmlns="http://www.cs.ucl.ac.uk/biobeacon/CMSL1.0#">

    <rdf:RDF xmlns:bqs="http://www.cellml.org/bqs/1.0#"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#">
        <!-- Meta Data goes here - Format is the same as in SBML/BIOMODELS models -->
        <!--  ...  -->
    </rdf:RDF>
    <phenomenon xmlns:sbml="http://www.sbml.org/sbml/level2">

        <phenomena id="Glucagon_Stimulated_Sig_Cascade" metaid="ph1">
            <!-- The main phenomena describe by the model -->
            <description>
                GLucagon hormonal stimulation of hepatocytes, and the
                resulting internal signaling cascade
            </description>
            <annotation>
                <!-- Link the phenomena to detection of Hormone Stimulus as listed in the Gene
                    Ontology -->
                <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
                    xmlns:dcterms="http://purl.org/dc/terms/"
                    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                    <rdf:Description rdf:about="#ph1">
                        <dc:isVersionOf>
                            <rdf:Bag>
                                <!-- Detection of Hormone Stimulus -->
                                <rdf:li
                                    rdf:resource="http://www.geneontology.org/#GO009720" />
                            </rdf:Bag>
                        </dc:isVersionOf>
                    </rdf:Description>
                </rdf:RDF>
            </annotation>
            <!-- Now list the compartments in which the above mentioned phenomena, described
                by the model, occurs-->
                <!-- First the Hepatocyte membrane, which we link to the Open Biomedical Ontologies term
                    hepatocyte and to the Gene Ontology term membrane -->
            <sbml:compartment id="Hepatocyte_Membrane"
                metaid="Hepatocyte_Membrane">
                <annotation>
                    <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
                        xmlns:dcterms="http://purl.org/dc/terms/"
                        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                        <rdf:Description
                            rdf:about="#Hepatocyte_Membrane">
                            <dc:isPartOf>
                                <rdf:Bag>
                                    <!--  hepatocyte  -->
                                    <rdf:li
                                        rdf:resource="http://obo.sourceforge.net/#OBO:0000182" />
                                </rdf:Bag>
                            </dc:isPartOf>
                            <dc:isVersionOf>
                                <rdf:Bag>
                                    <!--  membrane  -->
                                    <rdf:li
                                        rdf:resource="http://www.geneontology.org/#GO:0005886" />
                                </rdf:Bag>
```

```
                </dc:isVersionOf>
            </rdf:Description>
        </rdf:RDF>
    </annotation>
</sbml:compartment>
<!-- Next the hepatocyte cytoplasm, linked to the Open Biomedical Ontolgies term hepatocyte and
to the Gene Ontology cytoplasm -->
<sbml:compartment id="Hepatocyte_Cytoplasm"
    metaid="Hepatocyte_Cytoplasm">
    <annotation>
        <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:dcterms="http://purl.org/dc/terms/"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
            <rdf:Description
                rdf:about="#Hepatocyte_Cytoplasm">
                <dc:isPartOf>
                    <rdf:Bag>
                        <!--  hepatocyte  -->
                        <rdf:li
                            rdf:resource="http://obo.sourceforge.net/#OBO:0000182" />
                    </rdf:Bag>
                </dc:isPartOf>
                <dc:isVersionOf>
                    <rdf:Bag>
                        <!--  cytoplasm  -->
                        <rdf:li
                            rdf:resource="http://www.geneontology.org/#GO:0005737" />
                    </rdf:Bag>
                </dc:isVersionOf>
            </rdf:Description>
        </rdf:RDF>
    </annotation>
</sbml:compartment>
<!-- Now we list the aspects, the concrete measurables depicted by the model and which are
part of the depicted phenomena -->
<!-- First is G-Protein activation level, again linked to the corresponding term in the
Gene Ontology -->
<aspect id="G_Protein_Activation_Level"
    metaid="G_Protein_Activation_Level">
    <aspect_id>9234675</aspect_id>
    <annotation>
        <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:dcterms="http://purl.org/dc/terms/"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
            <rdf:Description
                rdf:about="#G_Protein_Activation_Level">
                <dc:isVersionOf>
                    <rdf:Bag>
                        <!--  G-Protein Activation  -->
                        <rdf:li
                            rdf:resource="http://www.geneontology.org/#GO:0004930" />
                    </rdf:Bag>
                </dc:isVersionOf>
                <dc:isVersionOf>
                    <rdf:Bag>
                        <!--  Activation level  -->
                        <rdf:li
                            rdf:resource="http://www.measurableproperties.org/#Activation_Level" />
                    </rdf:Bag>
                </dc:isVersionOf>
            </rdf:Description>
        </rdf:RDF>
    </annotation>
    <text_definition>
        Activation level of hepatocyte G Protein
    </text_definition>
    <description>
        The activation level of a G Protein activated by external hormone stimuli.
```

```
            </description>
        </aspect>
        <!-- Another aspect described by the model is the concentration of
            intracellular calcium. This time we define the term through a
            reference to the term 'Calcium' in the CHEBI ontology.  -->
        <sbml:specie id="Intracellular_Calcium"
            name="Intracellular Calcium Concentration"
            metaid="Instracellular_Calcium"
            compartment="Hepatocyte_Cytoplasm">
            <aspect_id>9234675</aspect_id>
            <annotation>
                <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
                    xmlns:dcterms="http://purl.org/dc/terms/"
                    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                    <rdf:Description
                        rdf:about="#Instracellular_Calcium">
                        <dc:isVersionOf>
                            <rdf:Bag>
                                <!--  Calcium  -->
                                <rdf:li
                                    rdf:resource="http://www.ebi.ac.uk/#CHEBI:22984" />
                            </rdf:Bag>
                        </dc:isVersionOf>
                        <dc:relation>
                            <rdf:Bag>
                                <!--  Concentration  -->
                                <rdf:li
                                    rdf:resource="http://www.measurableproperties.org/#Concentration" />
                            </rdf:Bag>
                        </dc:relation>
                    </rdf:Description>
                </rdf:RDF>
            </annotation>
            <text_definition>
                The concentration of intracellular Calcium in Hepatocytes
            </text_definition>
            <description>
                Calcium acts as an important second messenger.
                Changes in concentration and specifically
                oscillations occur as a result of hormonal stimulus, and
                in turn affect enzymatic activity within the cell
            </description>
        </sbml:specie>
    </phenomena>
</phenomenon>

<!-- The list of submodels of which this model is composed -->
<submodels>
    <submodel description_file="./G_Protein.xml"
        name="GProtein_activation" />
    <submodel description_file="./Calcium_cAMP.xml"
        name="PieceWise_Linear_Model_of_Calcium_Oscillations" />
</submodels>

<!-- This model has only one functionality, or possible usage. It can be used to
predict G-Protein activation levels and intracellular Calcium concentrations as a
function of blood Glucagon levels -->
<functionality
    name="G_Protein activation and Calcium levels as a function of hormone stimulus">
    <UsingAspects>
       <!-- One may refer here to aspects already defined in the phenomenon section, and
also define additional aspects. Aspects can be defined simply by referring to the
          aspect id in the parameter and aspect repository. -->
        <aspect id="Blood_Glucagon_Levels">
            <aspect_id>875446</aspect_id>
        </aspect>
    </UsingAspects>
    <ProvidingAspects>
```

```
            <aspect_ref id="Intracellular_Calcium_Level" />
            <aspect_ref id="G_Protein" />
        </ProvidingAspects>

        <!-- Here we list the functionalities of the submodels this model uses in order to implement
            its own functionality -->
        <using>
            <functionality
                functionality="G_Protein  activation level as a function of Hormone Stimuli and Calcium"
                model="G_Protein" />
            <functionality
                functionality="Calcium  concentration as a function of G_Protein activation level"
                model="PieceWise_Linear_Model_of_Calcium_Oscillations" />
        </using>

    <!-- The mathematical functionality term defines the functionality in precise mathematical terms.
        We specify that the output of the model contains timetrack (Function depicting how a
        variable changes through time) of G-Protein activation levels and of Calcium concentrations.
        The input is a timetrack of blood hormone levels -->
        <mathematical_functionality
            name="G_Protein activation level and  Calcium level vs. time as a function of
            hormone stimulation">

            <!-- We may specify the scheme that the model is implemented in - in this case,
                Ordinary Differential Equations -->
            <scheme composite="yes" type="differential_equations" />

            <!-- Here we define units that will be used later on. The specification
                is similar to that of CellML -->
            <units name="milli_mole_per_liter">
                <unit prefix="milli" units="mole" />
                <unit exponent="-1" units="litre" />
            </units>

            <!-- The variable type that appears here applies to all variables within
             this mathematical functionality, unless a variable
                is explicitely declared to be of another type. Thus
                we specify here that all the variables are of type
                timetrack -->
            <variable>
                <type name="timetrack" />
            </variable>

            <!-- List of output and input variables. Note that each variable
             is linked to the aspect it describes. -->
            <outputVars>
                <variable initial_value="0.0" id="G_Protein_Level"
                    units="pure_number">
                    <aspect_ref id="G_Protein" />
                </variable>
                <variable initial_value="0.0" id="Calcium_Level"
                    units="milli_mole_per_liter">
                    <aspect_ref id="Intracellular_Calcium_Level" />
                </variable>
            </outputVars>
            <inputVars>
                <variable initial_value="0.0" id="Hormone_Level"
                    units="milli_mole_per_liter" />


            </inputVars>

            <!-- Parameters are similar to input variables, the main difference being
            that their value remains fixed through out the course of the simulation -->
            <parameters>
                <parameter units="milli_mole_per_liter" id="IP3_ER"
                    name="IP3 concentration threshhold in Endoplasmic Reticulum">
                    <aspect_id>9865543</aspect_id>
```

```
        <type name="scalar" />
    </parameter>
    <parameter units="milli_mole_per_liter"
        id="Resting_GProtein"
        name="Resting concentration of inactive G-protein">
        <aspect_id>8754433</aspect_id>
        <type name="range" />
    </parameter>
</parameters>

<!-- Here the description of the interface provided by the
mathematical functionality ends, and we proceed to
describe how this functionality is actually implemented.
The orchestrator reads the 'implementation' section
and executes it in order to execute the model -->
<implementation>

    <!--  The model instances/components participating in the computation -->
    <!-- In this simple example we have one component per model,
    but one may specify mulitple components launched for the same model.
    For example, one may specify a composite multi-cellular model using
    multiple instances of the same cellular model -->
    <component model="GProtein_activation"
        id="G_Protein_Component" />
    <component
        model="PieceWise_Linear_Model_of_Calcium_Oscillations"
        id="Calcium_Component" />


      <!-- Next we map out the connections between the models. We first specify
      which components are connected to each other, and which functionalities
      and interfaces of each component are being used. Then
      we actually map output variables into input variables -->
      <!-- This connection is a 'horizontal' connection, it links
      sub-models on the same level -->
    <CMSL_connection id="GProtein_Calcium_Coupling"
        type="horizontal">
        <map_components>
            <mapped_component
                functionality="G_Protein and PLC activation level as a function of Calcium"
                mathematical_functionality="G_Protein and PLC activation levels vs. time
                as a function of Calcium"
                component="G_Protein_Component" id="G_Protein" />
            <mapped_component
                functionality="Calcium levels as a function of PLC activity levels"
                mathematical_functionality="Calcium levels as a function of PLC activity levels vs. time"
                component="Calcium_Component" id="Calcium" />
            <map_variable>
                <source mapped_component="G_Protein"
                    variable="PhosphoLipase" />
                <dest mapped_component="Calcium"
                    variable="PhosphoLipase" />
            </map_variable>
            <map_variable>
                <source mapped_component="Calcium"
                    variable="Calcium_Concentration" />
                <dest mapped_component="G_Protein"
                    variable="Calcium_Level" />
                <translator>
                    <scaling default_value="100.0"
                        name="Glucose_scaling" />
                </translator>
            </map_variable>
        </map_components>
    </CMSL_connection>
    <!-- This is a 'vertical' connections, wiring of the submodels variables to
    the 'global' model variables -->
    <CMSL_connection type="vertical">
```

```
                <map_components>
                    <mapped_component
                        functionality="G_Protein  activation level as a function of Calcium"
                        component="G_Protein_Component" id="G_Protein" />
                    <mapped_component id="Self" />
                    <map_variable>
                        <source mapped_component="G_Protein"
                            variable="G_Protein" />
                        <dest mapped_component="Self"
                            variable="G_Protein" />
                    </map_variable>
                </map_components>
            </CMSL_connection>
            <CMSL_connection type="vertical">
                <map_components>
                    <mapped_component
                        functionality="Calcium_Level as a  function of GProtein activation"
                        id="Calcium" component="Calcium_Component" />
                    <mapped_component id="Self" />
                    <map_variable>
                        <source mapped_component="Calcium"
                            variable="Calcium_Concentration" />
                        <dest mapped_component="Self"
                            variable="Calcium_Level" />
                    </map_variable>
                </map_components>
            </CMSL_connection>

            <!-- Now we specify what connector we use to actually implement the connection
                 between the two models. In this case we use a waveform relaxation connector -->
            <CMSL_connector id="GProtein_Calcium_Coupling">
                <integration_method name="Waveform_relaxation" />
                <implement_connection
                    connection="GProtein_Calcium_Coupling" />
            </CMSL_connector>

            <!-- This is the specification of the algorithm the orchestrator has to follow in order
                to execute this functionality. -->
            <sequence>
                <invoke name="step1">
                    <CMSL_connector
                        connector="GProtein_Calcium_Coupling" />
                </invoke>
            </sequence>
        </implementation>

    <!-- The last section in the mathematical functionality definition is the computational interface
     section. It describes the precise data format of the model's inputs and outputs -->
     <computational_interfaces>
         <implementation id="imp1">
                <engine name="C++_Orchestrator" version="0.1" />
                <!-- Here we specify additional data that the orchestrator may require in order
                    to execute the invocations, such as the computational interfaces to be used by the
                    connector -->
                <engine name="C++_Orchestrator" version="0.1" />
                <invoke step="step1">
                    <CMSL_connector
                        name="GProtein_Calcium_Coupling">
                        <use_interface mapped_component="G_Protein"
                            id="G_Protein_Interface" />
                        <use_interface mapped_component="Calcium"
                            id="Calcium_cAMP_Interface" />
                    </CMSL_connector>
                </invoke>

            </implementation>

        <!-- Here we define the computational interface for the (top level) model. In this case it is
```

```
             simply an output file -->
            <computational_interface id="G_Protein_Calcium_LZ"
                type="output_file">
                <subtype id="TimeTrack_Interface" implemention="imp1"/>
            </computational_interface>

             </computational_interfaces>
        </mathematical_functionality>
    </functionality>
</model>
```