VIRTUALIZATION IN SYSTEMS BIOLOGY: Metamodels and Modeling Languages for Semantic Data Integration

Magali Roux-Rouquié and Michel Soto

UMR 7606 - CNRS - Université Pierre et Marie Curie, LIP6
8 rue du capitaine Scott, 75 015 Paris, France
{magali.roux, michel.soto}@lip6.fr

Abstract. We examined the process of virtualization to deal with data intensive problems. Since data integration is a first-order priority in systems biology, we started developing a new method to manipulate data models through ordinary metadata transactions, i. e. by preserving the original data format stored in resources. After discussing why metamodels are made for, and the interplay of modeling languages in metamodel design, we presented a systemic metamodel-driven strategy to integrate semantically heterogeneous data.

1 Introduction

The process of virtualization has been defined as the mapping of an abstract data set to a virtual space according to three majors intertwined steps consisting of data selection for representing the problem space, assumptions definition to define the final virtual space, the mapping between the starting space and the final space through a metaphor [1]. Since that, virtualization has been extended to the management of distributed data, the main goal in this approach being to deal with data intensive problems [2].

The major features concern with the process of virtualization are:

- The preservation of data and knowledge in their actual format: the current physical reality and its putative evolution are not impacted at all by the virtualization process. This means that data and knowledge production can go their own way without any necessary change.

- The operationability: virtualization is not just abstraction, it allows to recursively transform the physical reality according to the lessons learned in virtual reality. In these respects, virtualization aims to actualize physical reality and, conversly, any change in physical reality has its counterpart in virtual reality.

Virtualization may be defined more formally by sets E and V corresponding, respectively to physical reality and virtual reality and the two functions M_E and M_V:

$$M_E : E \rightarrow V, \ M_V : V \rightarrow E$$

The functions M_E and M_V define the mapping rules between E and V. V is also named metaphor, which refers to a domain of knowledge. The choice of metaphor is driven by both the final goal and the actual possibility to define M_E and M_V.

The metaphor V can be homomorph or heteromorph. A metaphor V is homomorph when the initial E paradigm is used in the V design. Conversely, a metaphor V is heteromorph when a paradigm, different from the initial E one, is used to design V; as an important consequence, different metaphors can be selected to fit with different aims, according to a single physical reality.

Whatever M_E and M_V complexity, virtualization is a suitable process either to face heterogeneity problems or to shift from one domain of knowledge to more convenient representation as illustrated by the following examples.

Information technologies (IT) have been successful in using virtualization to deal with heterogeneity problems, to increase productivity of IT tools and to spread IT products to non IT-skilled users :

– Internet can be considered as the most famous and successful solution to a problem, thanks to virtualization. Because of protocol heterogeneity, existing computer communication architectures were not able to interoperate although they were built for the same purpose: exchanging data between distant computers. This is the process of virtualization that both allowed to integrate existing communication technologies and to preserve the future development; this was a key feature for internet success. To overcome this interoperability problem, a new protocol, named Internet Protocol (IP) and a new computer address format (IP address) were designed upward from the actual computer communication architectures; both IP and IP addresses were virtual in the sense they were not "natively" understood by any computer communication architectures. Mechanisms from IP and IP addresses to actual protocols and actual addresses (named respectively physical protocols and physical addresses) were designed to provide mapping. This was achevied without putting any constraints on future technological developments. Internet is an example of how an homomorph metaphor, the protocol/address paradigm, is of help in the virtualization process.

– Another example concerns the desktop metaphor enabling the use of computer hardware with a limited knowledge of the operating system (OS); in this case the metaphor was heteromorph since the desktop paradigm is very different from the computer hardware one.

- The third example concerns portability, which allows a software to be run on any microprocessor architecture without rewriting it (even partially) but just compiling it with the ad hoc compilator. Nevertheless, the use of compilators can not mask all differences between microprocessor architectures and there are always remaining portability problems. These increase dramatically when the target architecture is not know in advance as it occurs on internet. Virtualization was used to overcome this portability problem in the context of internet. A virtual microprocessor architecture, named virtual machine (or pseudo machine), and new high level programming language, named Java, was designed: Java compilators traduced Java written softwares in native instructions, named byte code, for the virtual machine and the virtual machine architecture closely mimiked actual microprocessor architectures. In these respects, the virtual machine is an isomorph metaphor with regards to the microprocessor architecture paradigm.

In the field of systems biology, the diversity of biological sources as well as experimental design and methodologies results in heavy heterogeneity, not only at the technological level but also at the semantic level; making data integration a major issue. In parallel, another challenge aims to simulate biological systems to predict their behavior; the ultimate goal being to understand not only their structure but also their dynamic [3].

To approach this problem, virtualization could be of great help. As it does not require any modification in the way of the data are produced, it preserves all accumulated experience and skills. Only mapping rules are concerned with the problem of data heterogeneity and further modifications in experimental approaches. Virtualization can be achieved by developing a metamodel-driven strategy to elicit a model upward from the current knowledge; the availability of such a metamodel for biological systems could be used as a grid for data integration. This needs to have a clear understanding of what a metamodel is made of, how it is designed and what it is doing for.

In these respects, metamodeling is not a final goal but an interface between data from the physical world and models in the virtual world. The mapping between the two worlds is iterative and model transformations are the operational side that misses single abstraction. As matter of fact, metamodeling is the junction that makes it possible to extend the database methodology to simulation thanks to the process of virtualization.

2 Metamodel

Since data integration is a first-order priority in systems biology, metamodel-driven strategies that are the foundation for data integration, should get much attention.

What is a metamodel ?

The methodology to provide a generic metadata abstracting and structuring all models into an integrated metadata repository consists into metamodeling. This means that a metamodel provides all concepts, properties, operations and relations between concepts necessary for designing any kind of models to be contained in it, at some level of abstraction and from some perspective. In these respects, a metamodel makes it possible to map multiple models into a single model by coalescing those elements identified as representing the same concepts.

In a metamodel, the notion of semantics is very important and reflects not only the need to model things in the real world (the signifier or the substance; for example, a molecular structure), but also the meaning that these things have to have for the purpose of the metamodel (the signified, the role in a particular context; for example, a molecular function).

To achieve a metamodel-driven integration, it is necessary to understand the meaning of data in all systems to be integrated: which data have the same meaning, which data are complementary and how they are related. Performing such a semantic analysis yields a metamodel for the types of data to be integrated; In these respects, one metamodel is a models integrator; conversely, the metamodel instances are models.

As a metamodel upward from the model layer, metamodeling deals with the full scope of paradigm translation, enabling the use of one model described into in one formalism to be transformed into a model in another formalism as far as each model is obeying well-formed rules, leading to possible model transformation and coupling.

In discipline-specific metamodeling (DSM), metamodels are the way to explicit the meaning of concepts in such a specific realm and to capture the relevant concepts. Among advantages, this approach allows to organize data without any modification of their structures. In addition, it makes it possible to check for the consistency of the multiple specifications as they do not conflict with one another and must be "in some sense" consistent. Also, it makes it easier to ensure model validation by comparing the computerized model to the model designed by the domain expert, for satisfactory range of accuracy.

From a technical point of view, a metamodel allows all local models and other metadata contained in it to be added, deleted, or modified through ordinary metadata transactions accounting for data and knowledge virtualization, in contrast to a fixed global data model.

In practice, the building of a metamodel will consider four levels:

- the information level 1 or data level, which consists in the basic facts to integrate;
- the data model at level 2, i. e. how the data are organized (for example, the model of a database consists in a special implementation of the metamodel);
- the metamodel (level 3) that describes and organizes concepts with a set of well-formed rules, to integrate all models from level 2;
- the language for metamodeling (metametamodeling, level 4) that use concepts and the relations defined in the metamodel, and may consist in, both, textual and/or graphic notations.

We present in figure 1, an example for a core language metamodel using a class diagram in the object-oriented paradigm: the left part describes classes accounting for the generic description of language elements called descriptor elements; the right part shows classes responsible for instantiation of generic elements and named instance elements. These two blocks are linked by binary associations setting the connection between the generic descriptor elements and the instance elements. They describe which elements of the instance level belong to which element of the generic level.

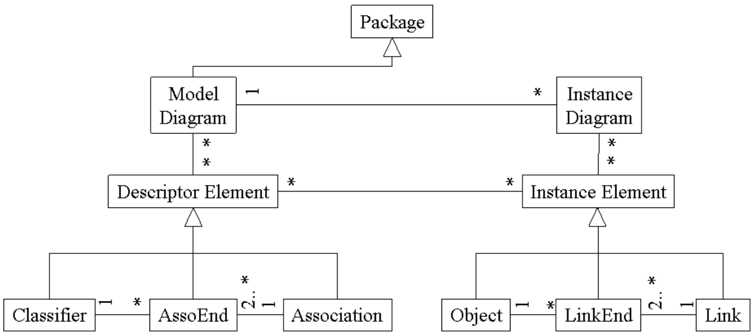

Fig. 1. A core metamodel for generic description and instantiation of language elements.

The Meta Object Facility (MOF) is a well known metamodel maintained by OMG [4]. It allows to create instances which are models, such as the Unified Modeling Language (UML) or the Common Wharehouse Metamodel (CWM).

A four levels modelization is used to metamodel UML: the UML model is defined with respect to the MOF, and the MOF is self-contained, i. e. it is used for self-definition. We summarized these four levels of metametamodelization (meta^{2}modelization), which have been defined by OMG:

- the M0 level contains specific information described at level 1 and is data;

- the M1 level represents instances of the UML metamodel;

- the level M2 corresponds to the UML metamodels described with the MOF, the UML metamodel being the language for creating UML models;

- the higher level, M3, corresponds to the MOF which is the language for designing metamodels. For example, the metaclass MOFClass has an instance which is the UMLClass; this recursive nature of the metamodel approach to the definition of the syntax of the UML (see below for details) is elegant.

3 Modeling languages

In addition to data integration, a metamodel is especially powerful when it is self-contained and does not require auxiliary means or external tools to specify itself; as such, it can be used as a true language to deal with as mentioned with the MOF which, not only allows the design of metamodels, but also allows its own design.

What is a modeling language ?

A modeling language is a language that contains all the elements with which a model can be described. It is a set of symbols and rules used to specify concepts and constructs for any kind of system; they may be textual and/or visual, structural and/or behavioral. Modeling languages are true languages and have syntax (the notation) and semantics (the meaning). Syntactic issues focus purely on the notational aspects of the language and modeling languages have to have a rigid syntax if they have to be further compiled.

In these respects, the structure of a modeling language is the following:

- An abstract syntax defines the different ways symbols may be combined to create well-formed models. Syntax defines the formal relations between the elements of the language; it deals with the form and the structure of the various expression of the language without any reference to their meaning.

- A graphical notation is the concrete syntax, the representation with well-formedness rules. For textual languages, the concrete syntax is a set of characters, the alphabet, characters are grouped into words and arranged into sentences according to precise grammar rules.

- A syntax mapping relates abstract syntax to concrete syntax and back; for example, the syntactic operator ”sum” is mapped to the graphical notation ”+”.

- A semantic domain defines the elements that are described by the abstract syntax; semantics considers the meaning of syntactically correct models: what to think, what to feel, what to do for natural language, the computer behavior for programming language.

- A semantic mapping gives the rules that map the syntax to constraints on things in the semantic domain, it gives the ”meaning” of the model according to the syntax. For example, the syntactic graphical operator ”+” in a arithmetic expression is mapped to the addition operator of arithmetic, so that the meaning of the expression 1+2 is to be the number 3, which is the sum of the two numbers.

With the standardization of the UML, the aim was to gather within a unique notation the best features of object-oriented languages. The usage of UML as a modelling language has an important impact and UML descriptions turn out to be abstractions used to capture important properties of the systems to be developed, notably in terms of static structure and dynamic behaviour. In these respects, UML is a true language and as such has syntax and semantics. The UML standard has chosen to use a metamodeling approach based on the very popular class diagram to characterize the abstract syntax of the language; nevertheless, the language is composed of an additional set of notations that may overlap. For example, the state diagram notation can be used to express the same information that could be express in terms of pre/post conditions on operations in class diagram, but there are other aspects of states diagrams that can not. Let us consider some examples of the semantics and syntax of the UML according to the dynamic aspects of the language.

- Statechart: from a semantic point of view, the UML statechart represents the behaviour of entities capable of dynamic behaviour by specifying their responses to the receipt of event instances. Typically, it is used for describing the behaviour of classes. From a syntactic point of view, a statechart is a graph that represents a state machine. States and various other types of vertices (pseudostates) in the state machine are rendered by appropriate state and pseudostate symbols, while transitions are generally rendered by directed arcs that interconnect them. A statechart maps into a StateMachine and a StateMachine is owned by a model element capable of dynamic behaviour.

- State: in UML, a state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or wait for some events. A state may be simple or composite it is used to model an ongoing activity that may be specified as a nested state machine or by a computational expression. A state is shown as a rectangle with rounded corners. A state may be subdivided into multiple compartments separated from each others by a horizontal line. Notably, internal transitions compartment holds a list of internal actions or activities that are performed while the element is in the state. A state symbol maps into a State. A composite state is decomposed into two or more concurrent substates or into mutually exclusive disjoint subtates; and any substate of a composite state can also be a composite state of either type. The notation of a composite state allows showing its internal state machine structure; concurrent states are shown by tiling the graphic region of the state using dashed lines to divide it into substates; in contrast, disjoint states are shown by showing a nested state diagram within the graphic region.

- Event: an event is a noteworthy occurrence; for practical purposes in states diagrams, it is an occurrence that triggers a state transition. Events may be of several kind. For example, the receipt of an explicit "signal" from one object to another results in a signal event instance; it is denoted by the signature of the event as a trigger on a transition. A signal can be declared using the <<signal>> keyword on a class symbol in a class diagram; such keyword is specified as <<stereotype>>.

- Transition: a transition is a relationship between two states indicating that an object in a first state will enter the second state and perform specific actions. It is notated as a solid line originating from the source state and terminated by an arrow on the target state. A transition string and the transition arrow that it labels together, map into a Transition and its attachment. A concurrent transition may have multiple sources states and target states. It represents synchronization and/or a splitting of control into concurrent threads. From the semantic point of view, a concurrent transition is enabled when all the sources states are occupied. After a compound transition fires, all the destination states are occupied. A concurrent transition includes a short heavy bar (a synchronization bar, which can represent synchronization, forking or both). A bar with multiple transition arrows leaving it maps into a fork pseudostate; conversely, a bar with multiple transition arrows entering it maps into a join pseudostate (figure 2).

These limited examples on behaviour specifications clearly point out of the respective parts relying UML syntax and semantics and their mapping. The very intuitive UML notation is even expressive enough to account for a large variety of situations; in these respects, UML customization can be achieved thanks to UML profiles that specify "standard elements" beyond those specified by the identified subset of the UML meta-model. (OMG Document: ad/99-03-10). Because it gathers the best features of object-oriented language, we think that UML fits all criteria for being developed as a profile in the realm of systems biology. This would benefit the major efforts achieved for the standardization of the language and the fine-tuning to systems biology would be achieve through exten-

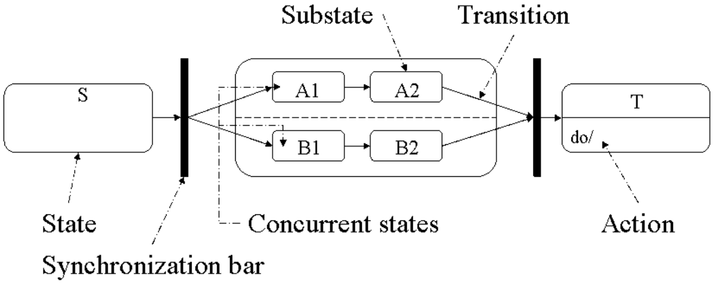

Fig. 2. States and concurrent transitions

sion capabilities. Accordingly, a limited number of well-known symbols would be necessary for deciphering the various states of one particular entity, most of the syntax and the semantics being defined by the language itself; in contrast, the dialect in systems biology being refined according to domain-specific ontology, metadata, etc.

The needs for language in Biology made of a limited number of symbol and a simple grammar, have been emphasized recently [5].To support this requirement, it was noticed that more than 75.000 articles were published since 1997 about the apoptosis death-programmed process without giving a clear understanding of it. It was suggested that poor data integration was accounting for such heavy difficulties. To encompass these bottlenecks, an international initiative was launched to set up the Systems Biology Markup language (SBML), a XML-based language, to facilitate data exchange and a Systems Biology Workbench (SBW) was developed for having heterogeneous application components to communicate [6]. A parallel project, BioSpice was using a Model Definition Language that was currently identical to SBML Level 2 [7]. Similarly the CellML project is an XML-based open standard for describing and exchanging models of cellular and subcellular processes [8]. In our hands, theses approaches mostly focus on technological integration and deal with data exchange showing variations between formatting whereas the semantic approaches was dealing with a limited number of topics, all necessary for building workflow models but not expressive enough to account for the large variety of biological phenomena and experimental approaches to depict. To fill this gap, graphical languages are being developed to achieve more detailed specifications. Cook [9] proposed a basic lexicon of icons and arrows for describing the function of complex biological systems. This approach was rooted in the work of Khon [10] that delineated large sets of molecular interactions maps. These initiatives and others [11, 12] have been synthesized to propose a standard graphical notation for specifying biological networks; introducing more structured specifications of biological systems in terms of expressiveness, consistency, extensibility, mathematical translation and software support [13].

Otherwise, ontologies [14] are under development to provide standardized vocabulary; they concern mostly the GO consortium that provides well-structured controlled terms on *Molecular Function*, *Biological Process*, *Cellular Component* [15], as well as related projects not to mention the BioProcess ontology that distinguish *logical* and *biochemical actions* to describe biochemical pathways [16].

At last, semantic mapping is an important part of the language structure in identifying important concepts and how these concepts fit together. This approach has been launched in molecular biology by the pioneer work of Paton [17] that identifies core question or concept, subordinate ideas that help explain or clarify the main concept, details, inferences and generalization that are related to each. This approach, which expressed biological knowledge as a society of graph, could be of great help in further topology mappings between models; for example, by mapping concepts from a vertex set of to a single vertex or from a path to a single edge.

4 Systemic metamodel and UML profile for systems biology

Of invaluable interests, all these initiatives can be merged within metamodel(s)in a virtualization perspective. Developing metamodeling approaches for systems biology require identifying primitive concepts, properties, operations and relations between concepts necessary for the specification of biological systems in terms of structure and behavior as well as the methodological approaches involved (i. e., genomics, transcriptomics, proteomics, etc.). In a more practical view, this can be approached through the metaphor of reactive systems to organize concepts and data in systems biology, just as the Windows makes use of the metaphor of desktop that was more familiar to office worker.

This issue can be achieved in the framework of the systemic paradigm [18], which allows to state that:

- a functional entity can be efficiently represented as the interface between an internal and an external environment in which it is evolving and on which it is acting;

- the behavior of this entity can be described as the trajectory of its states within a *Time*, *Space*, *Form* frame;

- events occurring from either internal and/or external environment may allow some changes in state variables and the consecutive firing of state transitions;

- all these changes can be modelled as the mapping between state description and process description.

It must be mentioned that the concept of *action* is central for virtualization [19], and the systemic paradigm in centered on. In these respects, viewing molecular entities as *processes*, interacting molecules as *communicating processes*, the change in interacting molecules as the *change in process states*, etc. [20], emphasizes the isomorphism between biological systems on the one hand, and reactive systems on the other hand, making real-time extensions of the UML available for customization to systems biology [21, 22]. This takes advantage of state diagrams to depict dynamic systems, which are grounded on the pioneer Harel's work on statecharts. This formalism was recently applied to biology by Kam et al. for the modeling of the immune systems [23]; nevertheless, Kam's approach missed a reference to an explicit systemic metamodel to organize data.

As defined in section 2, a metamodel must bring all elements to define a model; in these respects, a metamodel must acknowledge the mandatory requirements to integrate main data in systems biology, in terms of substances, constraints and processes (figure 3). In the systemic metamodel, this was achieved as follows:

- Substances consist in biological entities, which have a persistent identity. This must be clearly distinguished from the set of states taken by theses entities and that refer to their history. In our model, substances referred to any kind of biological entities, from organisms to molecules, and were arranged into specialization (Is-a) and composition (Has-a) hierarchies. In other words, substances were concerned with the identity (permanent) of the entities and not their states, which are transitory. As a consequence, the system was described with a limited number of classes (and relations) as the entities derived from these classes have several states. Substances were specified in the main class *Substance*, the child class *S_Molecular* has a specialized class *S_Protein* with a proteinId which stores accession numbers to database.

- The constraints (relationships) between system components constitute important aspects of living systems and most of the information we have on constraints in pathways comes from biochemistry chemical. But such changes are only half the story and our understanding of the functioning has to be completed with the spatial-temporal location of molecules in the cells as well as the properties attached to their three-dimensional behavior, i.e. all context effects. In our model, three kinds of class accounted for such Space-Time-Form constraints: (1)The *SpaceOccurrence* specified the position of any entity with regards to its external environment, (2) the *TimeOccurrence* referred to the age, time, period of any active entity (the time, the period this entity is functioning), (3) the *FormOccurrence* specified the functional isoform (if any) of the substance. The *FormOccurrence* was described as the set of *BioTransformation* (for example, phosphorylation, acetylation, etc.) that operated on the *BioSubstance*.

– Processes are represented - according to the systemic guidelines - as the state trajectories of entities functioning over time. Understanding processes requires the description, the modeling and the simulation of state trajectories of these entities. To achieve this goal, the concept of active object is very well adapted as active objects have their own behavior that can be described with subsets of state machines. In our metamodel, this allows us to delineate the elementary entity involved in process and named *Functional Unit* (*FUn*). A *FUn* has internal and external environment. Internal environment delineates the roles of *FUns* as *infraFunctionalUnit* (*infraFUn*): a functional entity has components that assume specific tasks to function; this corresponds to its internal environment; for example, the components of the general transcription factor TFIID that consist in TBP together with $8-12$ tightly bound subunits, constitute the internal environment of TFIID and play the role of "infra" functional units. In addition, *FUns* play two kinds of roles according to the external environment: they are *FUns* nesting *FUns*; in our metamodel, this role is named *supraFunctionalUnit* (*supraFUn*); furthermore, in their external environment, *FUns* have neighbor reacting entities, they referred to their *neighborFunctionalUnit* (*neighborFUn*); for example, their reactiveness can be assigned according to distances and/or domain affinity at the molecular level, or concentration at the population level. This can be modeled as messages passing between *FUns* and results into state transition, from the current state to a new state.

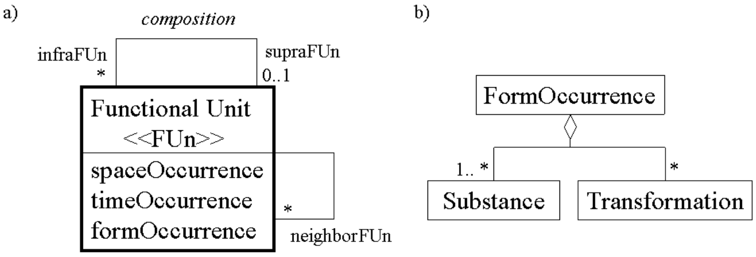

Fig. 3. Systemic metamodel: (a) the active upper class *FUn*, (b) the passive class *Substance*.

Summarizing the major features of the systemic metamodel needs to underscore the clear separation between structural and behavioral aspects with respect to the functional entities (*FUns*) which were modeled as processes using active objects, in contrast to substance that was modeled using passive objects. This was achieved in a perspective to extend the database methodology to the virtualization approaches.

Accordingly, the metamodel-driven strategy can be used to guide data integration as all concepts were being contained in it. Shortly, if we consider the

Microarray gene expression data model [24] that is detailed in the adopted specification of the OMG [formal/03-02-03], the *BioSequence* package, which contains representations of a DNA, RNA or protein sequence, could be integrated into the *Substance* package in the systems biology metamodel. Otherwise, limited part of a data model could be integrated to the metamodel; for example, the EntityLink.entity_id_(1,2) field of the Macromolecular Structure Specification [OMG Formal/02-05-01) that represents the entity ids of the two entities joined by a linkage, could be integrated at the metamodel level to specify the binding between FUns. The same approach could be achieved with both the *Bind-action-type* in the BIND database model [25] and the *Action-type* in [16], that can be integrated in the metamodel to specify action occurring in a particular state (figure 4).

Thus, a metamodel for systems biology would allow describing, in a common way, the data found in the large variety of physical sources by clarifying the hypotheses and the axioms that hold among concepts, as previously stressed in reference [18] concerning relationships between *Being* (OMB) [14] *Structural element* (EcoCyc)[26], *Cellular function* (GO) [15], *Cellular role* (YPD) [27] *Structural element* (EcoCyc) [26], *Processes* (GO) [15] *Pathways* (KEGG) [27], etc.

Because of the isomorphism between the systemic specifications of biological systems and the reactive systems used as a metaphor to drive the virtualization process, we consider the customization of the UML to systems biology, named SB-UML, instead of developing a new language. In order to assess the relevance of developing such UML extensions to systems biology (UML profile), we initiated the writing of Khon's molecular interaction maps into SB-UML. Shortly, we found SB-UML more expressive than the referenced graphical notation as it allows representing additional dynamic features. The figure 4 presents the complexation of protein A to protein B showing concurrent behavior of protein A and B until synchronization into complex AB is achieved. This approach emphasizes pattern occurrence allowing factored processes with, among important advantages, software reusability (to be published elsewhere).

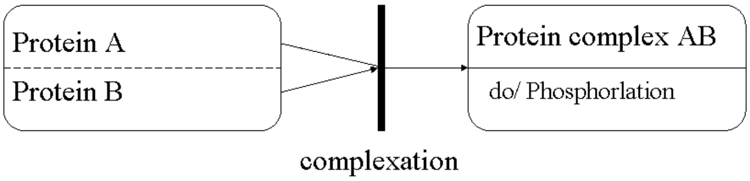

Fig. 4. Complexation is a synchronization process.

Details on the way one entity changes its respective states, take advantage of the reactive systems metaphor. Figure 5a shows aspartate transcarbamoylase, an instance of the the allosteric enzyme active class, containing an allosteric region and an enzymatic region, all stereotyped as <<FUns>>. The figure 5b gives the structure view of the enzyme that shows how the regulatory and enzymatic regions communicate with their environment through specific amino-acid residues symbolized as black squares. When the required interaction (signal) targets site-specific amino-acids, a transition is fired from the initial state to the final state. This corresponds to the changing from a free state to a bound state for the allosteric region and from an inactive state to an active state for the enzymatic one; both processes are concurrent and theses changes occur simultaneously. When the new states become occupied, the enzyme is allowed to perform carbamoyl transfer (figure 5c). As shown, extending UML to systems biology allows accounting for details that are no more mentioned in usual specifications because of some limits in language expressiveness.

5 Conclusion and perspectives

In this paper, we presented the metamodel-driven strategy as a key step in the virtualization process. As this approach requires a metaphor relevant to the final goals in the field of systems biology, we found that biological systems could be efficiently modeled as reactive systems within the systemic framework as previously reported [18]. This allowed us specifying any biological entity as an attribute vector depending of time, space and form variables, $\overrightarrow{v}\ (t, s, f)$, and it was achieved in the object-oriented paradigm using a systems biology extension of the UML (SB-UML). This aims to delineate a UML profile for systems biology, taking advantage of the UML expressiveness with regards to reactive systems.

In reactive systems, entities have their own thread of control and can behave concurrently with steps for synchronization. This shows isomorphism to biological entities which behave independently, although in a synchronized manner. The reference to the systemic framework was achieved according to the design of the attribute vector centered on the concept of form. This allowed to clearly distinguish the structure of biological entities from their behavior, as most of the structural data can be assigned to the substance passive class, whereas the form attribute of the active Functional Unit (FUn) class can be referred to the dynamic substance transformation according to time and space occurrences.

It must be strongly emphasized that a metamodel-driven strategy is not just setting a model upward from the other models but it has a core function for designing a new model from a former one or from the physical reality. This function is central to the process of virtualization, which realises the coupling of a physical reality with a constructed virtual reality. This is achieved by preserving the diversity of data, without any modification at the data model level and

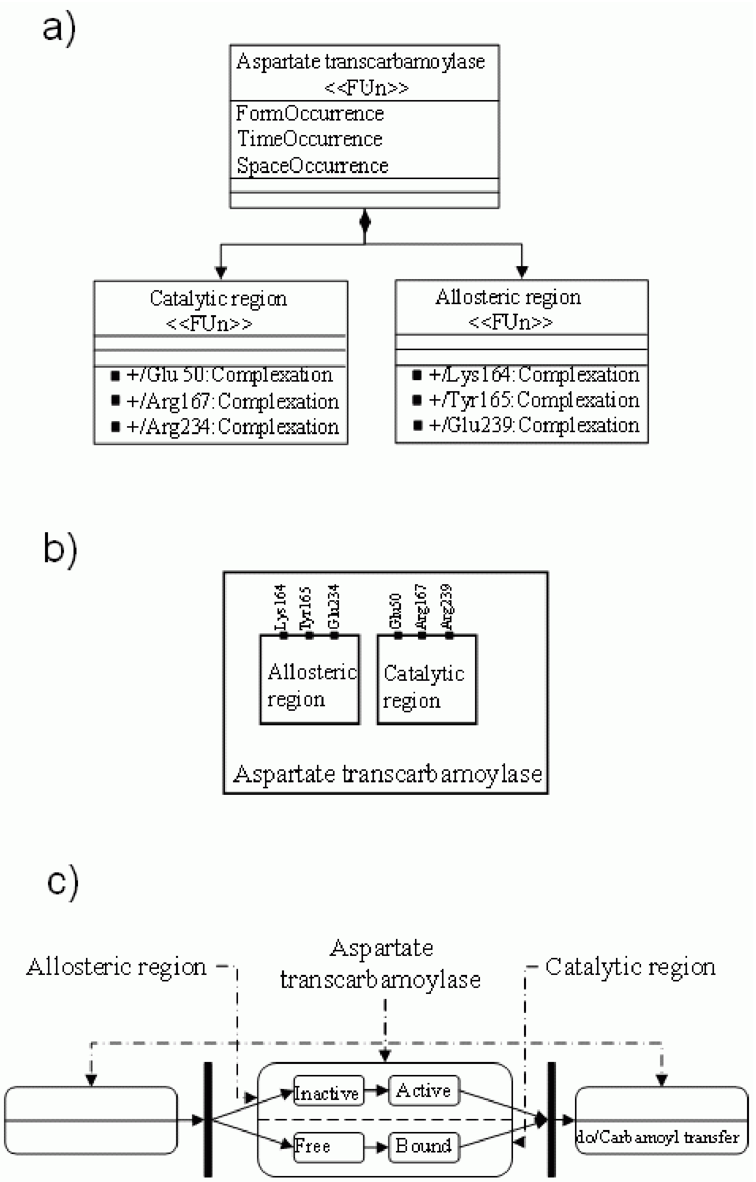

Fig. 5. Instantiation of the systemic metamodel: aspartate transcarbamoylase. (a) class diagram, (b) structure diagram, (c) state diagram.

without any hypothesis on their future improvements. As matter of fact, data evolution only impacts on the metamodel and the mapping rules between the physical reality and the virtual reality.

Virtualization leads to operationability, in the sense of actionability, since the virtual reality actualizes the physical reality; so that, any change in physical reality is reflected into virtual reality. In these respects, operationability is the main difference between abstraction and virtualization: abstraction aims to provide a general and synthetic point of view on reality, it does not provide any way to act on the abstracted reality. Conversely, virtualization neither aims to generalize nor to simplify: it aims to create a purpose-oriented virtual reality with action capability. In systems biology, the major aims for virtualization deal with heterogeneous data, data integration, analysis and simulation.

In these perspectives, our goal is to develop a method for virtualizing systems biology in any dimension of such systems i.e., data, process, experimental methodologies, modeling, etc. Part of the method, using SB-UML, will take advantage of the many efforts for translating UML diagrams into formal models suitable to carry out analysis on firm grounds [29]. Such transformations have different applications which may concern model checking to verify the global consistency, property verification at a low level of detail, simulation and prediction of properties, etc. Numerous works are ongoing in the fields of systems biology and we aim virtualization would help to integrate although preserving these different and complementary contributions.

Acknowledgments We thank Nicolas Caritey, Laurent Gaubert and Bénédicte Le Grand for helpful discussions.

References

[1] Spring, M. B., Jennings, M. C.: Virtual reality and abstract data: virtualizing information. Virtual Reality World. **1** (1) (1993), pp. c-m.

[2] Moore, R. W.: Integrating Data and Information Management. International Supercomputer Conference, June 22-25 (2004), Heidelberg (D).

[3] Auffray, C., Imbeau, S., Roux-Rouquié, M., Hood, L. (2003) C. R. Biologies. From functional genomics to systems biology. 326, 879-892.

[4] http://www.omg.org

[5] Franza, B. R.: From play to laws: Language in Biology. Sci STKE, pe9 (2004)

[6] Sauro, H., Hucka, M., Finney, A., Wellock, C., Bolouri, H., Doyle, J., Kitano,H.: Omics: A journal of integrative biology. **7** (2003) 355-372

[7] Garvey, TD., Lincoln, P., Pedersen, CJ., Martin, D., Johnson, M.: Omics: A journal of integrative biology. **7** (2003) 411-420

[8] http://www.cellml.org/public/specification/20021106/index.html

[9] Cook, D. L., Farley, J. F., Tapscott, S. J.: A basis for a visual language for describing, archiving and analyzing functional models of complex biological systems Genome Biology **2**(4) (2001) research0012.1-0012.10

[10] Kohn, K.W.: Molecular interaction map of the mammalian cell cycle control and DNA repair systems. Mol Biol Cell **10** (8) (1999) 2703-34. 3.

[11] Pirson, I., Fortemaison, N., jacobs, C., Dremier, S., Dumont, J., Maenhaut, C. The visual display of regulatory information and networks. Trends Cell Biol **10**(10) (2000):404-408.

[12] Maimon, R., Browning, S.: Diagramatic Notation and Computational Structure of Gene Networks. In: The Second International Conference on Systems Biology (2001). Pasadena.

[13] Kitano H.: A Graphical Notation for Biological Networks BIOSILICO **1** (2003) 169-176.

[14] Schulze-Kremer, S.: Ontologies for Molecular Biology, in Proc of 3rd Pacific Symposium on Biocomputing PSB98 (1998) 693-704.

[15] Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium, Nat Genet. **25** (2000) 25-29

[16] Rzhetsky, A., Koike, T., Kalachikov, S., Gomez, SM., Krauthammer, M., Kaplan, SH., Kra, P., Russo, JJ., Friedman, C.: A knowledge model for analysis and simulation of regulatory networks. Bioinformatics **16** (12) (2000) 1120-1128

[17] Paton, RC.: Diagrammatic Representations for Modelling Biological Knowledge. BioSystems **66** (2002) 43-53

[18] Roux-Rouquié, M., Le Moigne., JL.: The systemic paradigm and its relevance for modeling biological functions. C. R. Biologies, Special Issue : Model driven Acquisition **325** (2002) 419-430

[19] Soto., M.: Semantic approach of virtual worlds interoperability. In: Michael Capps (ed.): Proceedings of IEEE WET-ICE '97, Cambridge, MA, June 1997. IEEE Press.

[20] Regev, A., Shapiro, E.: Cells as computation. Nature **419** (6905) (2002) 343

[21] Roux-Rouquié, M., Renner, J., Sautejeau, G., Rosenthal-Sabroux, C.: Modeling Systems and Processes in Molecular Biology with active objects. In: Objects in bio- and chem-informatics (OiBCI02), OMG conference, Washington, USA (2002)

[22] Roux-Rouquié, M., Caritey, N., Gaubert, L., Rosenthal-Sabroux, C.: Using the Unified Modeling Language (UML) to guide systemic description of biological processes and systems Biosystems (2004), in press.

[23] Kam, N., Irun, R., Cohen, Harel, D.: The Immune System as a Reactive System: Modeling T Cell Activation With Statecharts. In: IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01) Stresa, Italy, (2001) september 05-07

[24] Spellman, Miller, PTM., Troup, C., Sarkans, U., Chevitz, S., Berhnart, D., Sherlock, G., Ball, C., Lepage, M., Swiatek, M., Marks, WL., Goncalves, J., Markel, S., Iordan, D., Shojatalab, M., Pizarro, A., White, J., Hubley, R., Deutsch, E., Senger, M., Aronow, BJ., Robinson, A., Bassett,

D., Stoeckert, CJ. Jr., Brazma, A.: Genome Biology **3** (9) (2002) research0046.1_0046.9

[25] G D Bader and C W V Hogue. BIND-a data specification for storing and describing biomolecular interactions, molecular complexes and pathways (2000) Bioinformatics 16, 465-477.

[26] Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A., Krummenacker, M.: EcoCyc: Encyclopedia of E. Coli Genes and Metabolism, Nucleic Acid Res. **27** (1999) 55-58.

[27] Hodges, P.E., McKee, A.H.Z., David, B.P., Payne, W.E., Garrels, J.I.: The Yeast Proteome Database (YPD): a Model for the Organization and Presentation of Genome-Wide Funtional Dat, Nucleic Acid Res, **27** (1999) 69-73.

[28] Kaneshisa, M.: A Database for Post-Genome Analysis, Trends genet. **13** (1997) 375-376.

[29] Korenblat, K., Priami, C.: Extraction of Pi-calculus specifications from a UML sequence and state diagrams. DEGAS IST-2001-32072, technical report (2003) #DIT-03-07.