A Categorical Model of Process Co-Simulation

Daniel-Cristian Crăciunean[1], Dimitris Karagiannis[2]
Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu, Sibiu, Romania[1]
Faculty of Computer Science, University of Vienna, Vienna, Austria[2]

Abstract*—A set of dynamic systems in which some entities undergo transformations, or receive certain services in successive phases, can be modeled by processes. The specification of a process consists of a description of the properties of this process as a mathematical object in a suitable modeling language. The language chosen for specifying a process should facilitate the writing of this specification in a very clear and simple form. This raises the need for the use of various types of formalisms that are faithful to the component subsystems of such a system and which are capable of mimicking their varied dynamics. Often in practice, the development of domain specific languages is used to provide building blocks adapted to the processes. Thus, the concept of multi-paradigm modeling arises which involves the combination of different types of models, the decomposition and composition of heterogeneous specified models as well as their simulation. Multi-paradigm modeling presents a variety of challenges such as coupling and transforming the models described in various formalisms, the relationship between models at different levels of abstraction, and the creation of metamodels to facilitate the rapid development of varied formalisms for model specification. The simulation can be seen as a set of state variables that evolve over time. Co-simulation is a synthesis of all simulations of the components of the system, coordinated and synchronized based on interactions between them. The theory of categories provides a framework for organizing and structuring formal systems in which heterogeneous information can be transferred, thus allowing for the building of rigorous cohesion bridges between heterogeneous components. This paper proposes a new model of co-simulation of processes based on the category theory.

***Keywords*—Process modeling; metamodel; modeling grammars; categorical grammars; category theory; categorical sketch; co-simulation, simulation**

I. Introduction

Contemporary systems are, in most cases, integrated from subsystems with complex structures and behaviors from the real or virtual world with various behaviors. Given the diversity of the components and the resulting complexity of the systems, simulation plays an essential role in all phases of system development and optimization. Simulation models at the system level can be developed to help analyze requirements, evaluate potential architectural solutions, and develop detailed design, implementation, and simulation specifications. These models aim at meeting specific objectives of each phase [2,12,14].

The concept of process is one of the possible methods of mathematical modeling of dynamic systems behavior. A process is a mathematical model that represents the behavior of a dynamic system that performs actions. Many systems, especially technological systems, can be modeled as Discrete Event System (DES) driven by discrete events evolving in relation to the occurrence of asynchronous events over time. These systems in which certain entities undergo transformations or receive certain services in successive phases can be modeled by processes.

The purpose of building a process that represents the behavior of a dynamic system is to facilitate the verification of system properties, simulation and system optimization. Therefore choosing the level of detail of the system's actions depends on the analyzed properties. Because a process does not take into account all the details, a behavior of an analyzed system can be represented by several processes that reflect either different degrees of detail of actions performed by a system or different points of view in relation to the intended purpose [12, 14].

Specifying a process consists of a description of the properties of this process in the form of a mathematical object. For this we need a proper modeling language. Therefore, the language chosen for specifying a process should facilitate the writing of this specification in a very clear and simple form. Thus, the need for the use of various types of formalisms that are faithful to the component subsystems of such a system and which are capable of mimicking their varied dynamics. Often in practice, the development of domain-specific languages, which provide building blocks adapted to DES models, is used in practice [4].

This heterogeneity of the systems inevitably implies the use of heterogeneous processes that provide through specific facilities a greater capacity to describe behaviors and interactions between subsystems than homogeneous processes. Thus, the concept of multi-paradigm modeling arises which involves the combination of different types of models, the decomposition and composition of heterogeneous specified models as well as their simulation.

Multi-paradigm modeling presents a variety of challenges such as coupling and transforming the models described in various formalisms, the relationship between models at different levels of abstraction, and the creation of metamodels to facilitate the rapid development of varied formalisms for model specification.

The simulation can be seen as a set of state variables that evolve over time. The variation space of the states of a simulation can be defined by two axes, a time axis, and a space axis. The objective of the simulation thus becomes, the calculation, ordered in time, of the state variable values. Co-simulation is a synthesis of all simulations of the components of the system, coordinated and synchronized based on interactions between them.

The theory of categories provides a framework for organizing and structuring formal systems in which heterogeneous information can be transferred, thus allowing for the building of rigorous cohesion bridges between heterogeneous components [8,15].

Multi-paradigm modeling involves, among other things, the transformation of structures from a given form into a form required to achieve this cohesion, which is often complicated. The category theory is based on the manipulation of structures consisting of objects and formal functions that coexist and work together as well as the preservation of these structures and their properties when they are transformed from one form into another through functors.

This paper proposes a new co-simulation model based on the category theory. In section 3 we present the construction of the simulation category, in section 4 we present the co-simulation model based on the categorical sketch and in section 4 we will see how the co-simulation category is built.

II. Theoretical Foundations and Notes

A category $\mathcal{C}$ is an algebra of formal functions in which the operation is the partial formal composition of functions [9,10]. The domains and codomains of functions form the set of objects of the category which we denote with ob($\mathcal{C}$) and the formal functions are the arcs of the category. We will denote these objects in uppercase A, B, .. X, Y, Z, ... The set of arcs between two objects $f{:}X\rightarrow Y$ will be denoted with $\mathcal{C}$ or Hom(X,Y).

So, a category is a construct structured from two types of atomic elements, formal functions that we call arrows and objects that are the domains and codomains of formal functions. This structure, completed with the composition operation of arrows, forms an edifice with a remarkable expressivity called the category. Because the functions are formal and the objects of the category could be formal, which implies great flexibility that is essential for modeling.

A functor is an application between two categories: $\phi{:}\mathcal{C}\rightarrow\mathcal{D}$ which maps objects to objects, arrows to arrows, and preserves the structure, i.e. transfers certain properties from one category to another [9,10].

One very important thing for modeling is that a functor can also be viewed as the image of a category in another category, that is, it can be viewed as a substructure consisting of objects and arrows taken together as one entity in a larger structure. These substructures are models of a category in another category. The set of categories together with the functors between them and the composition operation of functors form a category that is called the functors category and is written with Cat.

Between two functors $\phi{:}\mathcal{C}\rightarrow\mathcal{D}$ and $\psi{:}\mathcal{C}\rightarrow\mathcal{D}$, which have common domains and codomains, we define applications that take the image of ϕ into the image of ψ, respecting some naturality conditions in relation to the arrows in the two categories, which are called natural transformations [9,10].

The structure formed from the set of functors that have common domains and codomains, as objects, along with the set of natural transformations, as arrows, complemented by the composition of natural transformations is a category called the category of functors and natural transformations.

The essential difference between a graph and a category is the composition operation that exists in categories and does not exist in graphs [9,10]. But any graph $\mathcal{G}$ can be extended to a category called the free category generated by $\mathcal{G}$ which has as objects the nodes from $\mathcal{G}$, as arrows the arcs from $\mathcal{G}$ plus the identity arrows added to each object, and the composition operation is the concatenation of the paths from $\mathcal{G}$. In this way, any graph homomorphism naturally extends to a unique functor between the free categories generated by the two graphs. Note that not every functor between two free categories can be restricted to a graph homomorphism. With this remark, we will use the notion of functor even when the domain and/or codomain are graphs.

The image of a graph $\mathcal{P}$ in another graph $\mathcal{G}$ through a functor $D{:}\rightarrow\mathcal{G}$ is called the diagram of $\mathcal{P}$ in $\mathcal{G}$, and $\mathcal{P}$ is called the graph shape of the diagram D. Similarly, the diagram can also be defined if $\mathcal{G}$ is a category [9,10].

If we have a diagram $D{:}\rightarrow\mathcal{C}$ where $\mathcal{G}$ is a graph and $\mathcal{C}$ a category then a natural transformation from a constant diagram $\Delta_C{:}\mathcal{G}\rightarrow\mathcal{C}$ to D is a commutative cone with the vertex C and the base D [3,4, 9.10].

Among the cones we can define morphisms compatible with the natural transformations from the cone definition and so the set of cones together with these morphisms form the cone category generated by diagram D. The limit of a diagram D in a category $\mathcal{C}$ is a terminal element in the cone category generated by diagram D.

There are a series of particular limits, useful in modeling, such as the categorical product which is the limit of a discrete diagram or the limit of a cospan which is a pullback. The pullback for example is useful to characterize monomorphisms.

If we have a diagram $D{:}\rightarrow\mathcal{C}$ where $\mathcal{G}$ is a graph and $\mathcal{C}$ a category then a natural transformation from diagram D to a constant diagram $\Delta_C{:}\mathcal{G}\rightarrow\mathcal{C}$ is a commutative cocone with the vertex C and the base D [3,4 ,9,10].

Between cocones we can define morphisms compatible with the natural transformations from the definition of the cocones and so the set of cocones together with these morphisms form the category of cocones generated by the diagram D. The colimit of diagram D in a category $\mathcal{C}$ is an initial element in the cocone category generated by diagram D.

There are a series of particular colimits, useful in modeling, such as the disjoint union which is the colimit of a discrete diagram or the colimit of a span which is a pushout. The pushout for example is useful to characterize epimorphisms.

The notion of model exists also in logic. Its definition is based on the mathematical logic language. In the category theory a model can be specified by sketches. The major advantage of the sketches in modeling is that they can be defined by graphical notation for specifying visual modeling grammars.

Thus a sketch $\mathcal{S}$=($\mathcal{G}$, $\mathcal{D}$, $\mathcal{L}$, $\mathcal{K}$) consists of a graph $\mathcal{G}$ and three collections of diagrams, namely $\mathcal{D}$ which is a collection of commutative diagrams, $\mathcal{L}$ which is a collection of cones and $\mathcal{K}$ which is a collection of cocones [3,4,9,10].

The arrows of the graph $\mathcal{G}$ are sketch operators that can be implemented at the meta-metamodel level, possibly with small adjustments at the metamodel level. The three collections $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$ can be fully implemented at the meta-metamodel level, thus ensuring the syntactic correctness of any metamodel specified by a sketch.

A model of a sketch $\mathcal{S}$=($\mathcal{G}$, $\mathcal{D}$, $\mathcal{L}$, $\mathcal{K}$) is the image of the graph $\mathcal{G}$ through a functor M in the Set category that complies with all the conditions imposed by the collections $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$, i.e. it selects for each diagram in D a commutative diagram in Set, for each cone from $\mathcal{L}$ its limit and for each cocone from $\mathcal{K}$ its colimit [3,4,9,10].

III. The Simulation Category

The simulation consists in reproducing the dynamic behavior of a system in order to obtain conclusions about the behavior of the system. Simulation is very important for analyzing the behavior of complex systems. Simulation of a process in a certain formalism (such as PN, EPC, UML, BPMN) calculates the trace of the process execution in time represented by states, inputs and outputs. The simulation conclusions can be useful for determining the proper structure of the model, identifying the optimum values of the parameters, imitating system behavior, etc. [12,13,14]. In many cases, the model has predictive validity, i.e. it is able to predict the behavior of the system in the future [1,2].

Simulation of a system (behavior trace) can be described as a language on the set of states (inputs and outputs) in which each word represents a trajectory followed by states (inputs and outputs) of the system. Co-simulation composes the trajectories described by a set of components that interact. Interaction of components is made through incoming and outgoing ports that must be specified in the sketch that generates the model [2,11]. These will be associated to the nodes corresponding to the input and output constructs of the sketch.

Thus in [4], the sketch of the Medical Laser Manufacturing Systems (MLMS) metamodel specifies a model as a graph $\mathcal{G}$=(X,Γ,σ,θ) with imposed restrictions. The imposed restrictions lead to the sketch MLMS [4], L^1(MLMS)=($\mathcal{G}$,$\mathcal{D}$,$\mathcal{L}$,$\mathcal{K}$) where: $\mathcal{G}$ is the graph from Fig. 1, $\mathcal{D}$ is the set of commutative diagrams $\mathcal{D}$={D_1}, $\mathcal{L}$ is the set of cones $\mathcal{L}$={L_1,L_2,L_3,L_4} and $\mathcal{K}$ is the set of cocones $\mathcal{K}$={K_1,K_2,K_3}.

The graph of the sketch contains the nodes corresponding to the atomic concepts of the modeling language such as: set of input buffers for the primary components (xi), set of output buffers for finished products (xo) stations (xw), set of test stations (xt) and nodes corresponding to the associations between them (Fig. 1).

The diagram D_1 defines the function $\mu:\Gamma\rightarrow X\times X$ which will become a monomorphism provided that the pullback of μ with μ defined by cone L_3 is equal to Γ. The Cartesian product is the limit of the discrete diagram L_2. The function $\sigma_{wt}:\Gamma_{wt}\rightarrow X_w$ becomes a monomorphism provided that the pullback of σ_{wt} with σ_{wt} has to be Γ_{wt}, which is imposed by the cone L_4. The Cocone L_1 will require ω to become a terminal object in Set. The condition that the graph is connected is imposed by the limit of the cocone K_1 which will become a terminal element in Set. The cocones K_2 and K_3 serve to partition the concepts of the model [4].

To these nodes we add the node φ_i in the graph of the sketch representing the input interface and the node φ_o which represents the output interface represented by the input and output ports of the model. These will be associated with the input and output concepts by the arrows π_i and π_o (Fig. 1), which will have to become bijective functions in each model.

For the arrow π_i to become a surjective function, the pushout of π_i with π_i should be equal to x_i, i.e. the diagram from Fig. 2 has to become a pushout diagram in the Set category. In order for the arrow π_i to become an injective function, the pullback of π_i with π_i will have to be equal to φ_I, i.e. the diagram from Fig. 3 has to become a pullback diagram in the Set category.

Analogously, the arrow π_o becomes a surjective function if the pushout of π_o with π_o is equal to φ_o, i.e. the diagram from Fig. 4 becomes a pushout diagram is the Set category. And the arrow π_o becomes an injective function if the pullback of π_o with π_o will be equal to x_o, i.e. the diagram from Fig. 5 will become a pullback diagram in the Set category.

Therefore, the introduction of interfaces in the metamodel sketch implies in this case the addition of two more cones to the set of cones $\mathcal{L}$ according to Fig. 3 and Fig. 5 and adding two more cocones to the set $\mathcal{K}$ according to Fig. 2 and Fig. 4. Finally, it follows that the set of cones is $\mathcal{L}$={L_1,L_2,L_3,L_4,L_5,L_6} and the set of cocones is $\mathcal{K}$={K_1,K_2,K_3,K_4,K_5}.

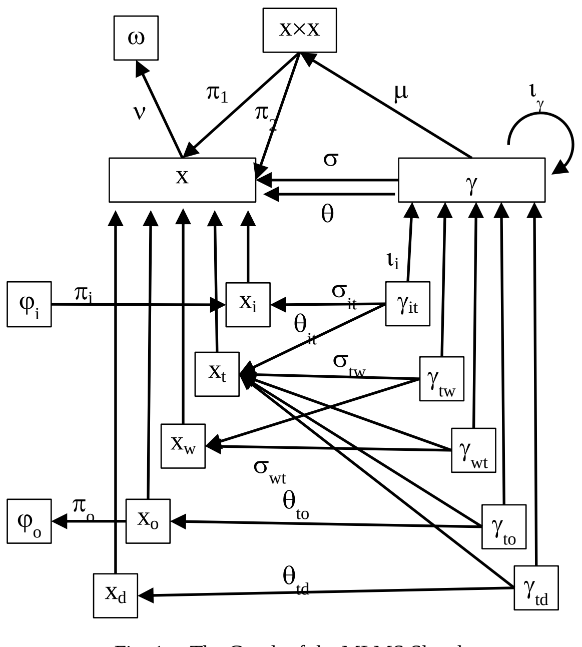


Fig. 1. The Graph of the MLMS Sketch.

$$\begin{tikzcd} \varphi_i \arrow[r, "\pi_i"] \arrow[d, "\pi_i"'] & X_i \arrow[d, "\iota_i"] \\ X_i \arrow[r, "\iota_i"'] & X_i \end{tikzcd}$$

Fig. 2. Pushout Diagram.

$$\begin{tikzcd} \varphi_i \arrow[r, "\iota_i"] \arrow[d, "\iota_i"'] & \varphi_i \arrow[d, "\pi_i"] \\ \varphi_i \arrow[r, "\pi_i"'] & X_i \end{tikzcd}$$

Fig. 3. Pullback Diagram.

$$\begin{tikzcd} X_o \arrow[r, "\pi_o"] \arrow[d, "\pi_o"'] & \varphi_o \arrow[d, "\iota_o"] \\ \varphi_o \arrow[r, "\iota_o"'] & \varphi_o \end{tikzcd}$$

Fig. 4. Pushout Diagram.

$$\begin{tikzcd} X_o \arrow[r, "\iota_o"] \arrow[d, "\iota_o"'] & X_i \arrow[d, "\pi_o"] \\ X_o \arrow[r, "\pi_o"'] & \varphi_i \end{tikzcd}$$

Fig. 5. Pullback Diagram.

A model of the sketch $\boldsymbol{\mathcal{S}}$ is a functor mapping the graph of the sketch in Set and all diagrams from D in commutative diagrams, all cones from L in cone limits and all cocones from K in cocone colimits.

The sketch $\boldsymbol{\mathcal{S}}$ reflects the relation between the abstract definition of a class of models and the concrete models specified by the sketch. Therefore, the sketch is the formal object that specifies the metamodel that in turn represents a class of models and contains all the semantics necessary to express the syntactic constraints of the entities in this class of models.

Therefore, a concrete model of the sketch L^1 is a functor $H^2:L^1\rightarrow$Set that associates to the classes (nodes), in the sketch L^1, set of extensions of these classes. The H^2 model associates to the nodes (classes) from L^1 with sets of extensions of these classes representing all types of objects that make up the model, as well as all types of relations that can be defined between the entities of the model and the arcs are the sketch operators. We will denote with $L^2=H^2(L^1)$ a model of the sketch L^1, i.e. a process constructed according to the grammatical rules imposed by the sketch L^1.

The behavior of a process is based on the state idea determined by the values of the attributes. The simulation begins with an event initializing the process with the data describing its initial state. Values of attributes that constitute the state of the system at one time produce events that trigger the execution of some actions of the process. Execution of these actions changes the state of the process and as a result determine again the execution of some actions [3,4].

So if we know the state of a process at a certain moment we can simulate the evolution of the process without knowing the history of the previous states that determined the current state. This means that the current state of a process concentrates the entire previous evolution of the process.

In our case a state of the process is represented by an instance of the model [3,4]. An instance of the L^2 model is a functor $\phi:L^2\rightarrow$Sets with the property that $\phi\circ H^2$ is a model of the sketch L^1, i.e. it complies with all the conditions imposed by the metamodel, and which associates to each set of classes from L^2 a set of instances, i.e. from each class one or more instances will be created.

If we have two instances $\phi,\psi:L^2\rightarrow$Set then we can define a natural transformation $\tau:\phi\rightarrow\psi$. The set of all instances together with all the natural transformations between them form a category that we call the process reaction category L^2 and we denote it with PRC. Thus, the evolution of the process from one state to another is represented by a natural transformation.

But the process has an initial state that in our case is an initial instance that we denote with $\mathfrak{I}_0$. We consider a subcategory of the PRC category, which we call the process simulation category (PSC) that has the objects: $Ob(PSC)=\{\mathfrak{I}_k|Hom(\mathfrak{I}_o,\mathfrak{I}_k)\neq\emptyset\}$, and the arrows are all arrows from the RPC category that have domains and codomains in Ob(PSC). The paths from the PSC category that have as a starting point object the instance $\mathfrak{I}_o$ represent the traces of the model in the simulation or execution process.

In the context of our model, simulation traces are sequences of instances that can be obtained by natural successive transformations from the initial instant. Each trace represents an alternative to executing the process. Therefore, if $\mathfrak{I}$=ob(PSC) then the set of simulation traces form a language $L(\mathfrak{I})\subseteq\mathfrak{I}^*$ defined as follows: $L(\mathfrak{I})=\{\ \mathfrak{I}_o\mathfrak{I}_1\ \ldots\ \mathfrak{I}_n\in\mathfrak{I}^*\ |\ \mathfrak{I}_k=\tau(\mathfrak{I}_{k-1})$ for $k\geq1$, and τ is a natural transformation and $\mathfrak{I}_o$ is the initial instance$\}$

IV. Categorical Model of Co-Simulation

The modeling of a large system involves the disassembly of the system into several real or virtual components from different domains integrated into a single model. Thus, the model is divided into several submodels, and each of these submodels requires the use of a certain formalism to specify the process in optimal conditions of fidelity, robustness and simplicity.

A process describes the behavior of a natural or artificial entity, real or virtual, under conditions imposed by a particular context. The context can also be represented by other processes that interact with the considered process, which leads to processes composed of several subprocesses. Both the composed process and its individual components are characterized by inputs, outputs, and internal states between which transitions are made that define how inputs and states cause outputs. The overall behavior of a process is therefore a composition of the individual behavior of its subprocesses.

In this type of hierarchical modeling, in which each component is independently specified, a model is a collection of models that in the simulation process must work together to achieve the common goal. This results in a co-simulation process that integrates several independent simulation subprocesses that synchronize to interact with each other. The concept of co-simulation involves subprocess coupling techniques to build the behavior of the integrated process.

Each subprocess has to provide through its ports its functions for other subprocesses involved in co-simulation. Also, the outputs of a subprocess influence the evolution of other subprocesses, and therefore the evolution of each process, although it seems independent, also depends on the evolution of the other processes.

Combining dependent and independent behavior of subprocesses is essential to the optimum process evolution, but can cause major problems if it is not done correctly. Subprocesses must be coupled through their inputs and outputs to reproduce the behavior of the integrated process [8]. Thus co-simulation of a process is the sum of the correlated simulations of the coupled subprocesses.

To coordinate the co-simulation, an orchestrator is required to control how components of the model are synchronized, translates and transfers data from subprocess outputs to inputs of other subprocesses, according to an appropriate co-simulation scenario. In our approach, the orchestrator is represented by a categorical sketch whose graph has as nodes the sketches generating the submodels involved and a series of association relations that will implement the interactions between the models. The other components of the sketch will impose submodel coupling conditions. We will call this sketch: the sketch of the co-simulation model or co-simulation sketch.

We will consider a co-simulation sketch $\mathcal{S}=(\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$, where the graph $\mathcal{G}$ has as nodes objects representing sketches of models and as arcs sketch operators. For example, the graph of a sketch with three models could be the one in Fig. 6.

This graph already implies several conditions on the co-simulation model, namely:

1) The models of the sketchs s1 and s3 do not interact directly with each other.

2) The models of the sketch s_1 interact directly only with the models from s_2, and the models from the sketch s_2 interact with those from the sketches s_1 and s_3.

3) The models from the sketch s_3 can interact with each other.

We can then introduce a set of other restrictions on the co-simulation model through the other components of the co-simulation sketch such as:

4) The models of the sketch s_3 should not contain loops, meaning there are no associations in which the source and targhet coincide.

This assumes that the coequalizer of the source and destination functions of the a_{33} association is the empty set, i.e. there is no arc in the model for which the destination and source coincide. In categorical terms, the coequalizer of σ_{33} and θ_{33} (Fig. 8) is the colimit of the diagram from Fig. 7, which we denote with K_1. This colimit will have to become, in the Set category, the empty set, so as in Fig. 3, i.e. the colimit of the void diagram that we denote with K_2 and which becomes the initial object in the Set category.

1) The commutative diagram from Fig. 9, which we denote with D_1, ensures the connection of model pairs two by two. Commutativity implies $\kappa_{12}\circ\kappa_{21}=id_{21}$, $\kappa_{21}\circ\kappa_{12}=id_{12}$, meaning that a_{12} and a_{21} are isomorphic. Similarly, a_{23} and a_{32} are also isomorphic.

2) Condition 5 does not assure us that there is only one pair of associations between two models. For this we have to put the condition that there is only one arc between any two models. Due to the isomorphism between a_{12} and a_{21}, it is sufficient to put the condition for one of them. We will put the condition for a_{12}.

$$\begin{tikzcd}
 & s_1 & \\
a_{12} \arrow[ur, "\sigma_{12}"] \arrow[dr, "\theta_{12}"] & & a_{21} \arrow[ul, "\theta_{21}"'] \arrow[dl, "\sigma_{21}"'] \\
 & s_2 & \\
a_{23} \arrow[ur, "\sigma_{23}"] \arrow[dr, "\theta_{13}"'] & & a_{32} \arrow[ul, "\theta_{32}"'] \arrow[dl, "\sigma_{32}"] \\
 & s_3 & \\
 & a_{33} \arrow[u, shift left, "\theta_{33}"'] \arrow[u, shift right, "\sigma_{33}"] &
\end{tikzcd}$$

Fig. 6. The Graph of a Sketch with Three Models.

$$\begin{tikzcd}
a_{33} \arrow[r, shift left, "\sigma"] \arrow[r, shift right, "\theta"'] & s_3
\end{tikzcd}$$

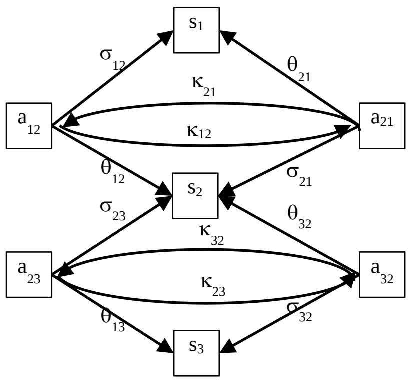
Fig. 7. Coequalizer Diagram.

$$\begin{tikzcd}
\varnothing \arrow[r, "\nu"] & A_{33} \arrow[r, shift left, "\sigma"] \arrow[r, shift right, "\theta"'] & S_3
\end{tikzcd}$$

Fig. 8. Coequalizer of σ_{33} and θ_{33}.

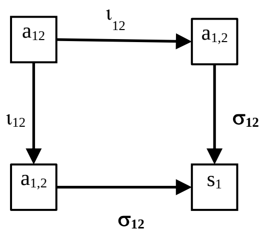
In the metamodel sketch we will have to impose the condition that σ_{12} and θ_{12} be isomorphisms. But, σ_{12} and θ_{12} are epimorphisms if and only if the diagrams from Fig. 12 and Fig. 13 are pushout diagrams, i.e. σ_{12} and θ_{12} are epimorphisms. We denote these two diagrams with K_3 and K_4. The functions σ_{12} and θ_{12} are monomorphisms if and only if the diagrams in Fig. 10 and Fig. 11 are pullback diagrams, i.e. σ_{12} and θ_{12} are monomorphisms. We denote with L_1 and L_2 these two limits.

For a_{23} and a_{32} the conditions are similar. So it also includes two more limits that we denote with L_3 and L_4 and two colimits that we denote with K_5 and K_6.



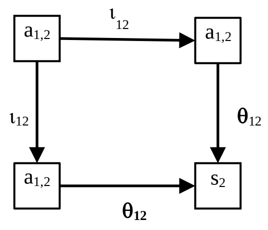
$$\begin{tikzcd}
 & s_1 & \\
a_{12} \arrow[ur, "\sigma_{12}"] \arrow[rr, bend right, "\kappa_{12}"'] \arrow[dr, "\theta_{12}"'] & & a_{21} \arrow[ul, "\theta_{21}"'] \arrow[ll, bend right, "\kappa_{21}"'] \arrow[dl, "\sigma_{21}"] \\
 & s_2 & \\
a_{23} \arrow[ur, "\sigma_{23}"] \arrow[rr, bend right, "\kappa_{23}"'] \arrow[dr, "\theta_{13}"'] & & a_{32} \arrow[ul, "\theta_{32}"'] \arrow[ll, bend right, "\kappa_{32}"'] \arrow[dl, "\sigma_{32}"] \\
 & s_3 &
\end{tikzcd}$$

Fig. 9. Commutative Diagram.



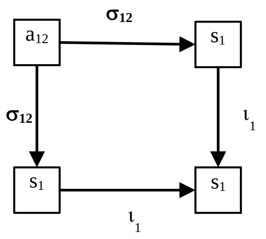
$$\begin{tikzcd}
a_{12} \arrow[r, "\iota_{12}"] \arrow[d, "\iota_{12}"'] & a_{1,2} \arrow[d, "\sigma_{12}"] \\
a_{1,2} \arrow[r, "\sigma_{12}"'] & s_1
\end{tikzcd}$$

Fig. 10. Pullback Diagram.



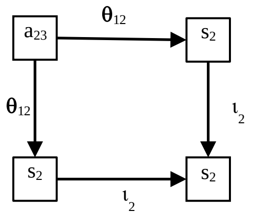
$$\begin{tikzcd}
a_{1,2} \arrow[r, "\iota_{12}"] \arrow[d, "\iota_{12}"'] & a_{1,2} \arrow[d, "\theta_{12}"] \\
a_{1,2} \arrow[r, "\theta_{12}"'] & s_2
\end{tikzcd}$$

Fig. 11. Pullback Diagram.



$$\begin{tikzcd}
a_{12} \arrow[r, "\sigma_{12}"] \arrow[d, "\sigma_{12}"'] & s_1 \arrow[d, "\iota_1"] \\
s_1 \arrow[r, "\iota_1"'] & s_1
\end{tikzcd}$$

Fig. 12. Pushout Diagram.



$$\begin{tikzcd}
a_{23} \arrow[r, "\theta_{12}"] \arrow[d, "\theta_{12}"'] & s_2 \arrow[d, "\iota_2"] \\
s_2 \arrow[r, "\iota_2"'] & s_2
\end{tikzcd}$$

Fig. 13. Pushout Diagram.

1) The models of the sketch s_3 form a connected graph. This assumes that the equivalence relation induced by the functions σ_{33} and θ_{33} on the set of models generated by the sketch s_3 will determine a single equivalence class on the set of models generated by the sketch s_3. For this, the diagram from Fig. 14 has to be a pushout diagram. That is, the span colimit determined by σ_{33} and θ_{33} is a terminal object in Set. We will denote with K_7 this diagram. We also need a terminal element from Set that is the limit of an empty diagram that we denote with L_5.

The graph of the co-simulation sketch is shown in Fig. 15.

The final sketch of the co-simulation model is $\mathcal{S}=(\mathcal{G},\mathcal{D},\mathcal{L},\mathcal{K})$ where: $\mathcal{G}$ is the graph from Fig. 4, $\mathcal{D}=\{D_1\}$, $\mathcal{L}=\{L_1,L_2,L_4,L_4,L_5\}$ and $\mathcal{K}=\{K_1,K_2,K_3,K_4,K_5,K_6,K_7\}$.

A model of a co-simulation sketch is a functor M:$\rightarrow$Set that associates to each node of type sketch from the co-simulation sketch a set of models according to the model sketch corresponding to the node, to each node of type association a set of functions between models and the arcs that are the sketch operators will be interpreted accordingly. Mapping will be done with respect to the restrictions imposed by the components $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$ of the co-simulation sketch.

The functions corresponding to the association nodes will translate the source model outputs into inputs of the destination model, thus ensuring communication between the models involved in the co-simulation.

Each model will therefore respect the conditions imposed by the sketch that generated it and the set of all the models that interact in the co-simulation process will respect the conditions imposed by the co-simulation sketch.

A co-simulation model of the sketch $\mathcal{S}$ from the example above could look like the one in Fig. 16 where we have denoted with hourglasses the models of the sketch s_1, with rectangles the models of the sketch s_2 and with diamonds the models of the sketch s_3. In this model we have 9 instances, 3 of each sketch. The arcs represented by lines in Fig. 16 are function types, images through the functor M of the association nodes in the co-simulation sketch. From them will create function instances that will do communication between the instances of the models. These functions, which we will call connection functions, map the outputs of a model to the inputs of another model. On the set of connection functions we can introduce the natural composing operation. The resulting construction generates a free category that has as objects the models and as arcs the connection functions. We will call this category: categorical model of co-simulation (CMCS).

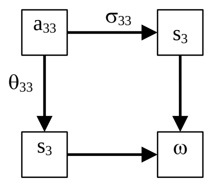


Fig. 14. Pushout Diagram.

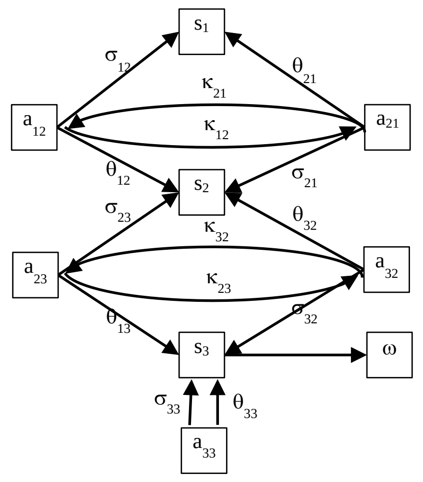


Fig. 15. The Graph of the Co-Simulation Sketch.

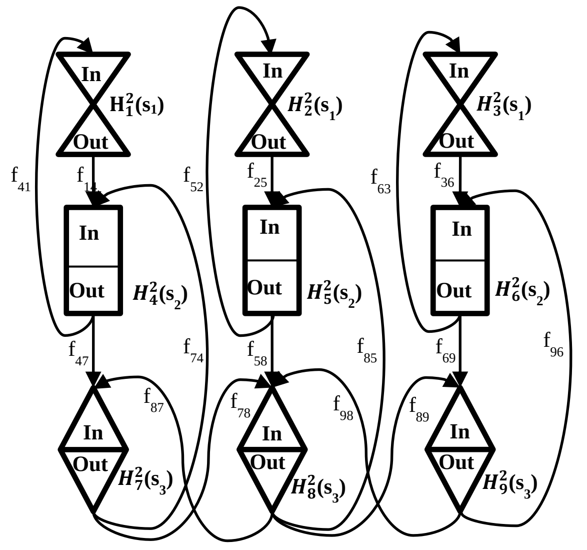


Fig. 16. Co-Simulation Model.

V. The Co-Simulation Category

The categorical model of co-simulation was created. The instances of each co-simulation model have as an interaction support an orchestrater provided by the co-simulation sketch. This sketch is a graphical specification of a co-simulation model provided by the category theory.

Let's see how an instance of the categorical model of co-simulation looks like. In the categorical model of co-simulation, the interaction is achieved through the connection functions. But in the co-simulation process, interaction occurs between models in certain represented states, as described in Section 3, through instance of component models. This means that interaction takes place between process simulation categories (PSC). It is natural that an instance of the categorical model of co-simulation will have to contain as objects process simulation categories (PSC) and as arrows connection functors between these categories that will be instances of the connection functions.

Therefore, an instance of the co-simulation model is a functor Φ:CMCS$\rightarrow$Set mapping each subprocess model M_i, i=1,2,..,n to the process simulation category corresponding to PSC_i, i=1,2,..,n, as we saw in section 3, and each connection function $f_{i,j}$ for i,j∈{1,2, .. n} to a connection functor $\phi_{i,j}$ for i,j∈{1,2, .., n} between the appropriate categories. Obviously, the image of this functor in Set (Fig. 17) can also be structured as a category in which the objects are simulation categories, the arrows are the connection functors and the composition is the composition of the functors. We will name this category the Reactive Category of Co-Simulation (RCCS).

We denote the set of objects ob(PSC_i)={$\Im_i^1$, $\Im_i^2$,…, $\Im_i^k$,…} for all i∈{1,2,..,n} and the arrows between two objects with $\Im_i^k$ and $\Im_i^l$ with τ_i^{kl}. A state $\mathfrak{J}_k$ of the co-simulation model is a tuple of the form $\mathfrak{J}_k$=($\Im_1^{k_1}$, $\Im_2^{k_2}$,…, $\Im_n^{k_n}$) where $\Im_i^{k_i}$∈ob(PSC_i) for all i∈{1,2,..,n}.

We first consider that processes are parallel and independent. Then there is a macrotransition between two states $\mathfrak{J}_k$=($\Im_1^{k_1}$, $\Im_2^{k_2}$,…, $\Im_n^{k_n}$) and $\mathfrak{J}_l$=($\Im_1^{l_1}$, $\Im_2^{l_2}$,…, $\Im_n^{l_n}$) if for every pair of instances $\Im_j^{k_j}$, $\Im_j^{l_j}$ there is a transformation $\tau_j^{kl} : \Im_j^{k_j} \rightarrow \Im_j^{l_j}$. But the evolutions in the categories of co-simulation processes are not independent, some macrotransitions are independent and may evolve as above, other instances wait for rendezvous with instances of other subprocesses for information exchange that require synchronization [13,15].

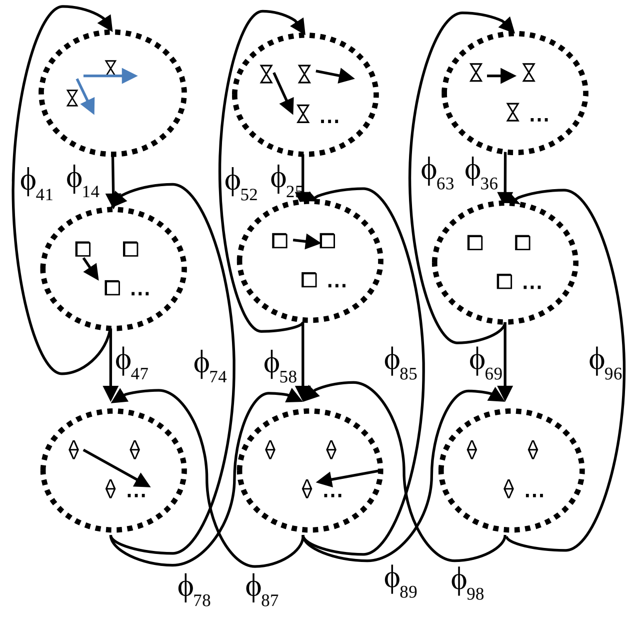


Fig. 17. Category of Co-Simulation.

For each process simulation category PSC_i ,i=1,2,..n we will construct a subcategory that we call the rendezvous category of the process RCP_i ,i=1,2,..n as follows: $ob(RCP_i)=\{\Im_i^k \in ob(PSC_i) \mid \exists$ j so that the execution of the transformation τ_i^{kj} depends on a rendezvous} and the arrows remain all from the PSC_i category which have the domains and codomains in $ob(RCP_i)$.

In this way, an instance of the co-simulation model becomes a functor Φ:CMCS$\rightarrow$Set that maps each subprocess model M_i, i=1,2,..,n to the rendezvous category of the process RCP_i, i=1,2,..,n, and each connection function $f_{p,q}$ for p,q$\in${1,2,.. n} to a connection functor $\phi_{p,q}$:$RCP_p \rightarrow RCP_q$ for p,q$\in${1,2,.., n} between the rendezvous categories of process as follows: $\phi_{p,q}(\Im_p^k)= \Im_q^l$ if and only if the instance $\Im_q^l$ needs as inputs the outputs of the instance $\Im_p^k$ to be able to evolve. Defining $\phi_{p,q}$ on arrows is implicit. Obviously, each resulting structure RCP_i, i=1,2,..,n is a category. We will call this category the category of co-simulation (CCS).

Then a macrotransition between two states $\boldsymbol{\Im}_k=(\Im_1^{k_1},\Im_2^{k_2},\ldots,\Im_n^{k_n})$ and $\boldsymbol{\Im}_l=(\Im_1^{l_1},\Im_2^{l_2},\ldots,\Im_n^{l_n})$ must meet the following conditions:

- For each pair of instances $\Im_j^{kj}$, $\Im_j^{lj}$ there is a transformation $\tau_j^{kl}: \Im_j^{kj} \rightarrow \Im_j^{lj}$. If $\Im_j^{kj} = \Im_j^{lj}$ then the transformation is the identity. In the simulation process this means that $\Im_j^{kj}$ is waiting for a rendezvous.
- For each pair p and q there is a transformation from $\phi_{p,q}(\Im_p^{kp})$ to $\Im_q^{mq}$ in the RCP_q category if and only if there is a transformation from $\Im_p^{kp}$ to $\phi_{q,p}(\Im_q^{mq})$ in the RCP_p category.

Condition ii) is necessary to avoid the deadlook situation in the co-simulation flow, i.e. a state in which each member of the tuple waits for another member to send its outputs to an instance that is not in the tuple or to receive inputs from another instance that is not in the tuple. Of course, each instance can execute the identity transformation for a number of steps but not for infinite without the process evolving. The condition ii) ensures that all instances will have the rendezvous they are waiting for, and there will be no deadlook. But obeying this condition is related to the definition of functors $\phi_{p,q}$ for p,q$\in${1,2,..,n}. For this, the functors $\phi_{p,q}$ and $\phi_{q,p}$ should be adjunct functors.

The functor $\phi_{p,q}$:$RCP_p\rightarrow RCP_q$ is the left adjunct of the functor $\phi_{q,p}$: $RCP_q\rightarrow RCP_p$ and $\phi_{q,p}$ is the right adjunct of $\phi_{p,p}$ and is denoted by $\phi_{p,q} \dashv \phi_{q,p}$ if and only if the set of arrows Hom($\phi_{p,q}$ -, -) and Hom(-,$\phi_{q,p}$ -) are naturally isomorphic as functors of two variables with values in Set. We denoted with – the place of a variable in the formula [9]. This means, in our case, that for each pair p and q there is a transformation from $\phi_{p,q}(\Im_p^{kp})$ to $\Im_q^{mq}$ in the RCP_q category if and only if there is a transformation from $\Im_p^{kp}$ to $\phi_{q,p}(\Im_q^{mq})$ in the RCP_p category, i.e. exactly the condition ii) from above.

But the two functors are adjuncts [9,10] if there is a natural transformation η_p: $id_p \rightarrow \phi_{q,p}\circ\phi_{q,p}$, where id_p is the identity functor in the RCP_p category and η_p is the adjunct unit, so for any objects $\Im_p^k$ from RCP_p and $\Im_q^l$ from RCP_q and any arrow $\tau_p^{kt}: \Im_p^k \rightarrow \phi_{q,p}(\Im_q^l)= \Im_p^t$, there is a unique arrow $\tau_q^{tl}: \phi_{p,q}(\Im_p^k) =\Im_q^t \rightarrow \Im_q^l$ so that the diagram in Fig. 18 commutes.

We also have the dual characterization that if two functors $\phi_{p,q}$ and $\phi_{q,p}$ have the property $\phi_{p,q}\dashv \phi_{q,p}$ then there is a natural transformation ε_q: $\phi_{p,q}\circ\phi_{q,p} \rightarrow id_q$ called adjunct county so that for any arrow τ_q^{tl}: $\phi_{p,q}(\Im_p^k)= \Im_q^t\rightarrow\Im_q^l$, there is a unique arrow $\tau_p^{kt}: \Im_p^k \rightarrow \phi_{q,p}(\Im_q^l)= \Im_p^t$ so that the diagram in Fig. 19 commutes.

Therefore, if the natural transformations η_p or ε_q exists with the above properties, the two functors are adjuncts. The natural transformation η_p provides a way to associate each arrow $\tau_p^{kt} : \Im_p^k \rightarrow \Im_p^k = \Im_p^t$ from the RCP_p category with an arrow τ_q^{tl} :$\phi_{p,q}$($\Im_p^k$)= $\Im_q^t \rightarrow \Im_q^l$ from the RCP_q category so that $\tau_p^{kt}=\phi_{q,p}(\tau_q^{tl})\ \eta_p$.

In our case, the natural transformation η_p:$id_p \rightarrow\phi_{p,q}\phi_{q,p}$ can be constructed taking into account the way the categories RCP_i i$\in${1,2,..,n} have been defined as subcategories of the process simulation categories PSC_i ,i$\in${1,2,..,n} in section 3. From this it follows that each Hom($\Im_p^0$, $\Im_p^k$) contains a natural transformation τ_p^{0k} for all k≥0. We will define every component η_p^k:$\Im_p^k\rightarrow\phi_{p,q}\phi_{q,p}(\Im_q^l)=\Im_p^t$ as follows: for each x$\in\Im_p^k$ we define $\eta_p^k(x)= \eta_p^k(\tau_p^{0k}(z)) =\tau_p^{0t}(z)$ where z$\in\Im_p^0$; x=$\tau_p^{0k}(z)$ and $\tau_p^{0k}\in$Hom($\Im_p^0, \Im_p^k$); $\tau_p^{0t}\in$Hom($\Im_p^0, \Im_p^t$). It is quite simple to prove that η_p defined as such respects the naturality property and thus is a natural transformation.

$$\begin{tikzcd}
 & RCP_p & RCP_q \\
\phi_{q,p}\phi_{p,q}(\Im_q^l) \arrow[dr, "\phi_{q,p}(\tau_q^{tl})"'] & \Im_p^k \arrow[l, "\eta_p"'] \arrow[r] \arrow[d, "\tau_p^{kt}"] & \phi_{p,q}(\Im_p^k) \arrow[d, "\exists!\,\tau_q^{tl}"] \arrow[ll, bend right, ""] \\
 & \phi_{q,p}(\Im_q^l) & \Im_q^l \arrow[l]
\end{tikzcd}$$

Fig. 18. Commutativ Diagram.

$$\begin{tikzcd}
RCP_p & RCP_q & \\
\Im_p^k \arrow[r] \arrow[d, "\exists!\,\tau_p^{kt}"'] & \phi_{p,q}(\Im_p^k) \arrow[d, "\tau_q^{tl}"] \arrow[dr, "\phi_{p,q}(\tau_p^{kt})"] & \\
\phi_{q,p}(\Im_q^l) \arrow[rr, bend right, ""] & \Im_q^l \arrow[l] & \phi_{p,q}\phi_{q,p}(\Im_q^l) \arrow[l, "\varepsilon_q"']
\end{tikzcd}$$

Fig. 19. Commutativ Diagram.

We observe that the component η_p^k of the natural transformation η_p exists if and only if the number of elements from $\mathfrak{I}_p^k$ is greater or equal to the number of elements in $\mathfrak{I}_p^t$. Similarly, the natural transformation $\varepsilon_q:\phi_{p,q}\circ\phi_{q,p}\to id_q$ can be constructed in the RCP_q category. If the natural transformations η_p^k and η_q^k exist for all $k\geq 0$ then the two functors $\phi_{p,q}$ and $\phi_{q,p}$ are the adjuncts and the condition ii) is fulfilled. Otherwise we will have to try all the variants admitted by the model in question to define the functors $\phi_{p,q}$ and $\phi_{q,p}$ to obtain two adjunct functors. If there is no such option we will have to make changes at the model level. In principle, it should be observed that for each object $\mathfrak{I}_p^k$ from the RCP_p category, the functor $Hom(\mathfrak{I}_p^k,\phi_{q,p}\text{-}):RCP_q\to Set$ has a universal element [9]. But we will deal with this problem in a future paper.

VI. Conclusion

Multi-formalism modeling aims to facilitate the use of more modeling formalisms in certain situations where it is necessary to compose heterogeneous models, thus allowing experts from different disciplines to collaborate more effectively in the development of increasingly complex systems. This approach involves specifying processes through different modeling grammars.

Many times the solution to this challenge is the development of grammars specific to the component subprocesses starting from a common meta-metamodel that facilitates their coupling in a co-simulation process that allows the study of the overall system behavior. There is a gap of remarkable results in the field of modular coupling, of simulators in dynamic structure scenarios at the state level [2]. The concept of categorical modeling method along with the MM-DSL [3,4,5,6,7] language facilitates this approach. The present paper proposes a co-simulation category (CCS) as a model for the co-simulation state space. The categorical sketch for co-simulation can be specified in a graphical language provided by the category theory for specifying the syntax of the co-simulation model.

This facilitates the separation of the model specification from the execution algorithms. Universal constructs offered by the category theory can be implemented as universal algorithms and mechanisms [6,7] at the meta-metamodel level and used in each model for process coupling. This type of algorithms reduce the complexity of syntax specification for coupling and provides support for domain specific modeling and distributed execution. Thus, the universal constructs from the category theory can be seen as a collection of tools for specifying and structuring the dynamic coupling of processes.

Synchronization can be elegantly modeled by adjunct functors. Composition of adjuncts is also an adjunct [9]. We have seen in section 5 the determination of the adjunct unit, if it exist it can be relatively simple. We will have to find general criteria for characterizing the situations in which these adjuncts exist. This problem as well as the problem of finding practical and efficient algorithms for determining synchronization adjuncts will be addressed in a future paper.

References

[1] Bernard P. Zeigler, Alexandre Muzy, Ernesto Kofman, Theory of Modeling and Simulation Discrete Event and Iterative System Computational Foundations - Academic Press 2019 , ISBN: 978-0-12-813370-5.

[2] Claudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, Hans Vangheluwe - Co-simulation: State of the art, - ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2016.

[3] Daniel C. Crăciunean, Dimitris Karagiannis, 2018, Categorical Modeling Method of Intelligent WorkFlow. In: Groza A., Prasath R. (eds) Mining Intelligence and Knowledge Exploration. MIKE 2018. Lecture Notes in Computer Science, vol 11308. Springer, Cham.

[4] Daniel C. Craciunean, Categorical Grammars for Processes Modeling" International Journal of Advanced Computer Science and Applications(IJACSA), 10(1), 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100105

[5] Dimitris Karagiannis, Robert Andrei Buchmann 2018, A Proposal for Deploying Hybrid Knowledge Bases: the ADOxx-to-GraphDB Interoperability Case, Published in HICSS, DOI:10.24251/hicss.2018.510.

[6] Dimitris Karagiannis, N. Visic, 2011. Next Generation of Modelling Platforms. Perspectives in Business Informatics Research 10th International Conference, BIR 2011 Riga, Latvia, October 6-8, 2011 Proceedings

[7] Dimitris Karagiannis, Heinrich C. Mayr, John Mylopoulos, 2016. Domain-Specific Conceptual Modeling Concepts, Methods and Tools. Springer International Publishing Switzerland 2016.

[8] Vagner, D., Spivak, D. I., & Lerman, E. M., 2015. Algebras of open dynamical systems on the operad of wiring diagrams. *Theory and Applications of Categories*, *30*, 1793-1822, Nov 30 2015.

[9] Michael Barr, Charles Wells, 2012. Category Theory For Computing Science, Reprints in Theory and Applications of Categories, No. 22, 2012.

[10] R. F. C. Walters, 2006. Categories and Computer Science, Cambridge Texts in Computer Science, Edited by D. J. Cooke, Loughborough University, 2006.

[11] Robin Milner, 2009. The Space and Motion of Communicating Agents, Cambridge University Press, 2009. ISBN 978-0-521-73833-0

[12] Weske, Mathias, 2012. Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer 2012, ISBN 978-3-642-28615-5, pp. I-XV, 1-403.

[13] Winskel Glynn, 2009. Topics in Concurrency, Lecture Notes, April 2009.

[14] Wil M.P. van der Aalst, 2011. Process Mining Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag Berlin Heidelberg 2011.

[15] Diskin Z., König H., Lawford M., 2018. Multiple Model Synchronization with Multiary Delta Lenses. In: Russo A., Schürr A. (eds) Fundamental Approaches to Software Engineering. FASE 2018. Lecture Notes in Computer Science, vol 10802. Springer, Cham.