A Monadic Co-simulation Model for Cyber-physical Production Systems

Daniel-Cristian Crăciunean
Computer Science and Electrical Engineering Department
Lucian Blaga University of Sibiu
Sibiu, Romania

***Abstract*—The production flexibility required by the new industrial revolutions is largely based on heterogeneous Cyber-Physical Production Systems models that cooperate with each other to perform complex tasks. To accomplish tasks at an acceptable pace, CPPSs should be based on appropriate cooperation mechanisms. To this end a CPPS must be able to provide services in the form of functionalities to other CPPSs, and also to use functionalities of other CPPSs. The cooperation of two CPPS systems is done by co-simulating the two models that allow the partial or total access of the functionalities of one system, by the other system. Requests from one CPPS to another CPPS create connection moments of the two models that can only be performed in certain states of the two models. Also, the answers to these requests create connections between the two models in other subsequent states. Optimal aggregation of the behaviors of the two models, by co-simulation, is essential because otherwise it can lead to very long waiting times and can cause major problems if not done correctly. We will see in this paper that the behavior of such a simulation model can be represented by a category, and the co-simulation of two models can be defined by a monad determined by two adjoint functors between the simulation categories of the two models.**

Keywords—Models; metamodels; co-simulation; adjoint functors; monads; cyber-physical production systems

I. Introduction

The new industrial revolutions ("Industry 4.0", "Industry 5.0"), respond to the needs of individualization of production by the widespread introduction of Cyber-Physical Production Systems (CPPS) as central elements in making production flexible at all levels. In essence, CPPSs are complex systems made up of heterogeneous entities and subsystems that cooperate with each other depending on the context in which they evolve at all levels of production.

CPPSs are composable systems, that is, they are systems of systems. The composition operation is determined by the interactions between the subsystems in all the phases of the life cycle of the production process. CPPSs are of overwhelming importance in the production process because they implement the interaction between physical and cybernetic components in distributed networks and therefore represent the fusion between the real and the virtual world as a whole.

In this context, it is obvious the importance of approaches for the optimal design and implementation of CPPS. Modeling and simulation are the most common approaches in the process of designing these systems. Modeling makes it possible to simulate and analyze production processes as well as make decisions before the actual construction of the manufacturing line [7] [8]. Also, after the construction of the production line, the models can be used for its optimization and diagnosis.

The primary artifact in the process of designing and implementing a CPPS is the model. In such a model the emphasis falls, most of the time, on the interaction and cooperation between the heterogeneous components of the system, and not on the internal functionality of these components [20]. Therefore, classical approaches, from systems theory, cannot satisfactorily respond to these interaction modeling requirements [9,22]. Our approach is motivated by the finding of a deficit of sufficiently strong mathematical mechanisms to be an adequate support for such models [14]. This vacuum of remarkable results is even more accentuated in the field of coupling simulators in dynamic structure scenarios at the level of states [21].

We will see further in this paper that category theory, which, unfortunately, is not used enough in modeling, provides all the ingredients needed to specify such models. Thus, in section 2 we will specify a categorical model of a CPPS, in section 3 we will specify the behavior of a model through a category we call Model Simulation Category, and in section 4 we will see that the co-simulation of two models can be defined by a monad determined by two adjoint functors between the simulation categories of the two models. The paper ends with conclusions.

II. Categorical Modeling of CPPS

An essential phase of the process of developing a model is the conceptualization of the domain [4]. A model is an artifact recognized by an observer as an abstract representation of a real system [1]. This artifact is a syntactic and semantic specification of the real system from the point of view pursued.

The conceptualization of the modeling domain begins with the identification of the generic atomic concepts of the domain that will represent classes of entities in the modeling domain [5]. The state of these atomic concepts is generally specified by the values of some associated attributes. This process continues with the specification of the interaction rules of these atomic concepts in models. We will consider in the following that the models are structured as graphs that have as nodes atomic concepts and as arcs interactions between these atomic concepts. The elementary components of the modeling domain will become atomic concepts in models. The behavior of atomic concepts is dependent on the state in which they are and the context in which they evolve, i.e., the graph structure in which they are integrated.

The specification of a model involves two specification mechanisms, namely a model specification paradigm through the hierarchical assembly of components and a mathematical model that mimics the behavior of the system. The first mechanism must reflect the static dimension of the model and the second the behavioral dimension of it. Therefore, both the syntax and the semantics of a model are characterized by two dimensions, namely a static dimension and a behavioral dimension.

A. The Static Dimension of CPPS Models

To specify the syntax of the static dimension of CPPS models we use the categorical sketch (Fig. 1.) which is defined as a tuple $\mathcal{S}$=($\mathcal{G}$, $\mathcal{C}(\mathcal{S})$), where $\mathcal{G}$ is a graph with typed nodes and arcs, and $\mathcal{C}(\mathcal{S})$ is a set of constraints on the elements of the graph that specify in categorical terms the conditions that a model must meet. Thus, the categorical sketch becomes a metamodel of the static dimension of our model.

The atomic concepts identified in the conceptualization phase of the domain become nodes of the graph $\mathcal{G}$ of this sketch. Thus, the metamodel that specifies the static dimension of CPPS models is the graph $\mathcal{G}$, which respects the constraints $\mathcal{C}(\mathcal{S})$, and which has as nodes the atomic concepts and as arcs the rules of aggregation of these concepts in models.

Therefore, in the metamodel $\mathcal{S}$ of the static dimension of CPPS models, $\mathcal{G}$ is a graph whose nodes specify the types of atomic concepts they represent and whose arcs specifies the types of connections between these concepts.

The ($\mathcal{S}$) component is defined at the metamodel level by a diagram predicate signature [13,15], which maps a set of predicates to a set of special graphs called shape graph arity. These special graphs, called shape graph arity, are then mapped by a set of functors, called diagrams, to the components of the graph of the sketch and therefore receive the types of these components. The set of models of the sketch $\mathcal{S}$ is represented by the set of functors defined on the graph $\mathcal{G}$ of the sketch $\mathcal{S}$, with values in the category of sets and functions (Set), and which respects the constraints defined by $\mathcal{C}(\mathcal{S})$ (Fig. 1.). These models are, from a syntactic point of view, also graphs that inherit from the graph of the sketch the type of components.

The components, of the graph $\mathcal{G}$ of the sketch $\mathcal{S}$, are endowed with attributes that are mapped to data domains. The semantics of the static dimension of a model is given by the graph structure of the model and by the values of the attributes of the atomic components.

In our approach, the set of static models represents the set of states in the behavioral model. Such a state is characterized by the graph structure of the model and the values of the attributes.

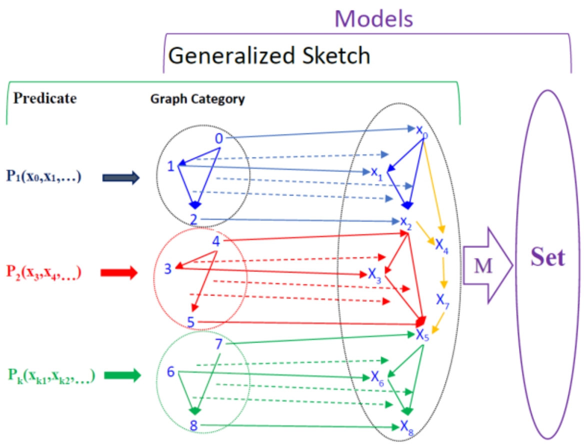

Fig. 1. Generalized Sketch and Models.

B. The Behavioral Dimension of CPPS Models

The behavior of a CPPS is shaped by the multitude of states in which the CPPS model can be found and by the multitude of transitions that ensure the transition from one state to another. In our approach, the set of states is represented by the set of static models and the set of transitions by a set of transformations of the model that we have called behavioral rules. We associated these behavioral rules with the dynamic components of the model, such as workstation or transport machine components.

We defined the behavioral rules by a tuple τ=(p,σ), formed by a graph transformation p and a behavioral action σ. A behavioral action σ, is a mathematical function that modifies the values of the attributes associated with the graph components in a certain area of the graph structure of the static model. The role of the graph transformation p is to locate the definition area of the behavioral action and to transform the graph structure from this area of the graph model into another graph structure, provided that the resulting graph structure is also a static model of the categorical sketch.

The syntax of a behavioral rule τ, is composed of the syntax of the two components, namely the syntax of the graph transformation p, and the syntax of the behavioral action σ.

Syntactically a graph transformation is a tuple p = (L, R), where L is a source graph to be transformed and R is the target graph in which the graph L is to be transformed. To define the semantics of a graph transformation we used the categorical mechanism called double-pushout (DPO) [16,17,18,19]. The DPO mechanism defines a graph transformation as a span p=(L$\leftarrow$K$\rightarrow$R), where K is a common part of the graphs L and R, and therefore there are two total inclusion morphisms p_L:K$\rightarrow$L and p_R:K$\rightarrow$R.

We defined the syntax of a behavioral action by a signature of a behavioral rule that maps a mathematical function to the components of the L and R graphs of the graph transformation. Thus, a signature of a behavioral rule is a tuple σ=(p,C_L,Act,C_R,α), where p is a graph transformation, Act is a mathematical function, C_L is a precondition, C_R is a postcondition and α:{C_L,Act,C_R}$\rightarrow$p is an application that maps the parameters of the Act action and of the preconditions C_L and C_R to the nodes of the graphs L and R.

The L and R components of the graph transformation play in this case the role of shape graphs for the signature of the behavioral action and will be mapped, by means of diagrams, to the graph components of the categorical sketch, mechanism by which they will receive the types of these components. The semantics of behavioral actions will be defined by mathematical functions that recalculate the values of the attributes associated with the graph components L and R.

The application of a behavioral rule is preceded by the finding of a total morphism, called a match, from shape graph L to the static model. If a match is found, the precondition C_L is checked. If this precondition is verified, the corresponding graph transformation is performed using the DPO algorithm and then the Act action is executed. Finally, the postcondition is checked. If the post-condition is not verified, the behavioral rule cannot be applied and, therefore, we will have to cancel all the effects produced by the partial execution of the behavioral rule. As we can see, a behavioral rule must be applied by an indivisible instruction. We must also note that the behavioral rule must be endogenous, i.e., it must transform a static model of the categorical sketch into another static model of the same sketch. It is obvious that the application of behavioral rules can be done in parallel as long as this application is not conflicting [Ehrig2015].

III. Model Simulation Category

The problem of CPPS optimization is a complex, multi-objective problem, which involves dynamic behavior accompanied by elements of their uncertainty, and therefore cannot be solved by optimization models from classical mathematics. The saving solution to this problem is simulation, which allows the study of the behavior of these complex systems in order to optimize them and eliminate deficiencies from the design phase.

We will understand by simulation the process of imitating the behavior of a materialized system through a multitude of possible trajectories through which it can evolve. Therefore, the simulation can be described as a language on the set of states through which the system can pass. We define an execution of the behavioral model as a word of this language.

In our approach, a state of the behavioral model is represented by a static model of the categorical sketch $\boldsymbol{\mathcal{S}}=(\boldsymbol{\mathcal{G}},\boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{S}}))$, and the transition between these states is made by the behavioral rules. A static model of the sketch $\boldsymbol{\mathcal{S}}$ is the image of a functor $\mathfrak{I}:\boldsymbol{\mathcal{S}}\rightarrow$Set that maps the nodes of the typed graph $\boldsymbol{\mathcal{G}}$ of the sketch to sets of components in the category Set and the arcs of the graph $\boldsymbol{\mathcal{G}}$ to functions in Set. Next, we will call the image of the sketch $\boldsymbol{\mathcal{S}}$ by a functor $\mathfrak{I}$, instance and we will also denote it with $\mathfrak{I}$. The component $\boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{S}})$ of the sketch imposes restrictions on the image $\mathfrak{I}(\boldsymbol{\mathcal{S}})$ in set, which will also be respected. Each node x of the graph $\boldsymbol{\mathcal{G}}$ represents a type of component of the system, and $\mathfrak{I}$(x) is a set of components of type x. We will consider that these components also contain values for the associated attributes. Also, each arc of the graph $\boldsymbol{\mathcal{G}}$, represents an operator of the sketch and is mapped to a function in the Set category. These functions are constitutive elements of the constraints ($\boldsymbol{\mathcal{S}}$).

If we have an instance $\mathfrak{I}_1$ and a behavioral rule τ, which transforms $\mathfrak{I}_1$ into $\mathfrak{I}_2$, then τ must be endogenous, i.e., $\mathfrak{I}_2$, must also be a static model of the same sketch $\boldsymbol{\mathcal{S}}$, this is a condition that must be respected by any behavioral rule. We will denote this by $\mathfrak{I}_1 \overset{\tau}{\rightarrow} \mathfrak{I}_2$.

With these notations, an execution of a behavioral model, in an initial state $\mathfrak{I}_0$, is a chain of behavioral rules: $\mathfrak{I}_0 \overset{\tau^0}{\rightarrow} \mathfrak{I}_1 \overset{\tau^1}{\rightarrow} \dots \mathfrak{I}_n \overset{\tau^n}{\rightarrow} \dots$, where $\mathfrak{I}_k$, k≥0 are instances and τ^k, k≥0 are behavioral rules.

We can introduce a partial operation of composing two behavioral rules. If we have two behavioral rules $\tau_1=(p_1,\sigma_1)$ and $\tau_2=(p_2,\sigma_2)$ then the behavioral rule $\tau=\tau_1\circ\tau_2$ is defined by composing the components $(p_1\circ p_2, \sigma_1\circ\sigma_2)$. Since, σ_1 and σ_2 are mathematical functions, their composition is specific to these functions. For graph transformations, we have a theoretical result that demonstrates that any two sequential graph transformations can be composed into a single equivalent graph transformation that accumulates the effect of both transformations.

Therefore, the set of behavioral rules is endowed with a composition operation. Obviously, this composition operation is associative and to the set of behavioral rules we can add the identity transformation which does nothing. It follows from these considerations that the set of instances together with the set of behavioral rules form a category that we call category of instances and behavioral rules (CIBR). The objects of this category are instances and its arrows are behavioral rules.

Now we can define an execution of a behavioral model, also as the image of the category Ω: $0 \overset{\alpha_0}{\rightarrow} 1 \overset{\alpha_1}{\rightarrow} \dots k \overset{\alpha_k}{\rightarrow} \dots$ through a functor $\Gamma:\Omega\rightarrow$CIBR that specifies the evolution of the model over time: Φ (i)= $\mathfrak{I}_i$ for all i≥0; Φ (α_i)= τ^i for all i≥0 as can be seen in Fig. 2. Any execution of a model starting from an initial state $\mathfrak{I}_0$ produces a simulation trace that is a word in a language defined on the set ob(CIBR) of objects of the CIBR category, through the set of behavioral rules.

Thus, if $\mathfrak{I}$=ob(CIBR) then the set of simulation traces, relative to the initial state $\mathfrak{I}_0$, form a language $L(\mathfrak{I})\subseteq \mathfrak{I}^*$ defined as follows: $L(\mathfrak{I})=\{ \mathfrak{I}_o\mathfrak{I}_1\dots \mathfrak{I}_n \in\mathfrak{I}^* | \mathfrak{I}_k=\tau(\mathfrak{I}_j)$ for k≥1, k≥j≥0 and τ is a behavioral rule$\}$.

If $L(\mathfrak{I})$ is the language of simulation traces over the vocabulary $\mathfrak{I}$ =ob(CIBR), and $\beta= \mathfrak{I}_o\mathfrak{I}_1\dots \mathfrak{I}_n\in L(\mathfrak{I})$, we will denote by $\nu(\beta)$ the set of instances involved in this trace, $\nu(\beta)=\{ \mathfrak{I}_o,\mathfrak{I}_1,\dots, \mathfrak{I}_n\}$.

We now define a subcategory of the CIBR category, which we call model simulation category (MSC), which has the objects ob(MSC)= $\{\mathfrak{X}|(\exists)\beta\in L(\mathfrak{I})$ so that $\mathfrak{X}\in\nu(\beta)\}$, and as arrows the coresponding behavioral rules. The PSC category contains all the trajectories on which the CPPS model can evolve. We denote this category by MSC($\mathfrak{I}$).

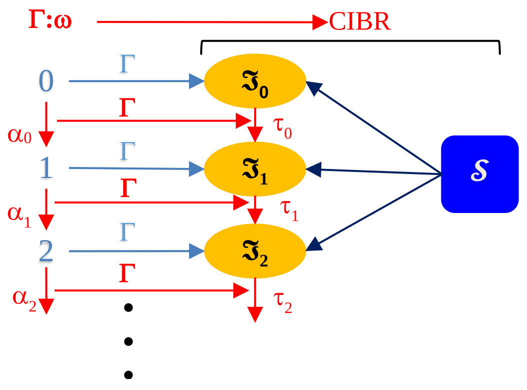

Fig. 2. Execution of a Behavioral Model.

IV. Monadic Co-simulation

The essential role of CPPS in the production processes is to make these processes more flexible in order to realize a lot of individualized products or in small batches, adapted to the customers' requirements. These CPPS must also cover many aspects of the production process. Therefore, there cannot be a single CPPS, which can deal with this variety of requirements. In order to carry out the tasks at an acceptable pace, the CPPS must contain adequate cooperation mechanisms.

A CPPS is a fundamental component of a smart factory and therefore must monitor the status of all participants in the production process and be able to automatically react to any event in order to achieve the objectives. For this purpose, a CPPS must be able to offer services, in the form of functionalities to other CPPSs, and also to use functionalities of other CPPSs.

The cooperation of two CPPS systems is done by co-simulating the two models, which allows partial or total access to the functionalities of one system, by the other system. The co-simulation control is realized by an orchestrator that coordinates the system components according to a co-simulation scenario.

However, in order to achieve a good cooperation, each CPPS must know about the functionalities of the other CPPSs with which it collaborates and how to extract and use the knowledge from the specifications of these CPPSs with which it cooperates in order to achieve the goal. This means that it must encapsulate knowledge about the state and evolution of the other CPPS with which it could cooperate.

The requests of a CPPS to another CPPS create moments of connection of the two models that can only be carried out in certain states of the two models. Also, the answers to these requests create connections between the two models between other subsequent states. This type of interaction between two models is realized between the simulation categories of the two models through co-simulation. The optimal aggregation of the behaviors of the two models, through co-simulation, is essential because, otherwise, it can lead to very long waiting times and can cause major problems if it is not done correctly [25]. We will further introduce a categorical modeling model of this type of connection based on the notion of monad specified by the adjunction of two functors.

On the set $\mathfrak{I}$=ob(MSC), we introduce a relation $\preccurlyeq$, defined as follows, $\mathfrak{I}_i \preccurlyeq \mathfrak{I}_j$, if and only if in the category MSC($\mathfrak{I}$), there is a path from $\mathfrak{I}_i$ to $\mathfrak{I}_j$, i.e. if $\mathfrak{I}_j$ can be obtained from $\mathfrak{I}_i$ through the successive application of a series of behavioral rules. Obviously, this relation is a partial preorder relation and therefore the set $\mathfrak{I}$ is a preordered set in reference to this relation. We observe that if L($\mathfrak{I}$) is the language of simulation traces over the vocabulary $\mathfrak{I}$=ob(MSC), then all simulation traces $\beta=\mathfrak{I}_o\mathfrak{I}_1\dots\mathfrak{I}_n \in$ L($\mathfrak{I}$), respect the condition: $\mathfrak{I}_i \preccurlyeq \mathfrak{I}_j$ if and only if $i \leq j$.

We now consider two model simulation categories MSC_1, MSC_2 with $\mathfrak{I}$=ob(MSC_1), $\mathfrak{L}$= ob(MSC_2). The connection points between the states of the two models, relative to the first model, can be specified by two monotone functions; a request function $\varphi:\mathfrak{I}\to\mathfrak{L}$ and a response function $\psi:\mathfrak{L}\to\mathfrak{I}$. To ensure a correct collaboration between the two models, without deadlock situations in the co-simulation flow, it is necessary that any two simulation traces $\beta_1=\mathfrak{I}_o\mathfrak{I}_1\dots\mathfrak{I}_n\in L(\mathfrak{I})$ and $\beta_2=\mathfrak{L}_o\mathfrak{L}_1\dots\mathfrak{L}_m\in L(\mathfrak{L})$, where $m,n\geq 0$, to respect the conditions: for each pair of states ($\mathfrak{I}_i$, $\mathfrak{L}_j$), $\mathfrak{I}_i\in\nu(\beta_1)$, $\mathfrak{L}_j\in\nu(\beta_2)$ with the property, $\mathfrak{L}_j=\varphi(\mathfrak{I}_i)$, there is a state $\mathfrak{L}_k\in\nu(\beta_2)$, such that $\varphi(\mathfrak{I}_i)\preccurlyeq\mathfrak{L}_k$, and $\mathfrak{I}_i\preccurlyeq\psi(\mathfrak{L}_k)$. This condition is necessary to avoid deadlock situations, in which the two simulations block each other, because each is waiting for a response from the other.

This type of cooperation between two models in the simulation process requires, therefore, that the functions φ and ψ form a Galois connection [24]. A Galois connection between the sets $\mathfrak{I}$ and $\mathfrak{L}$ is a pair of monotone mappings $\varphi:\to\mathfrak{L}$ and $\psi:\mathfrak{L}\to\mathfrak{I}$ with the property: $\varphi(\mathfrak{I}_i)\preccurlyeq\mathfrak{L}_j$ if and only if $\mathfrak{I}_i\preccurlyeq\psi(\mathfrak{L}_j)$ where $\mathfrak{I}_i\in\mathfrak{I}$ and $\mathfrak{L}j\in\mathfrak{L}$. The two applications, φ and ψ are the left adjunct and, respectively, the right adjunct of the Galois connection.

A theoretical result [23,24] tells us that the condition in the above definition of the Galois connection can be replaced by the condition: $\mathfrak{I}_i\leq\psi(\varphi(\mathfrak{I}_i))$ and $\varphi(\psi(\mathfrak{L}_j))\leq\mathfrak{L}_j$. We will see, next, that this condition has a generalization in category theory and is called adjunction.

Since MSC_1 and MSC_2 are categories, we can replace the applications φ and ψ with two adjunct functors, $\Phi:MSC_1\to MSC_2$ and $\Psi:MSC_2\to MSC_1$, and thus obtain an adjunction between two functors, where Φ and Ψ are the left adjunct and, respectively, the right adjunct of the adjunction. The adjunction of two functors Φ and Ψ is denoted by $\Phi\dashv\Psi$, where Φ is the left adjunct and Ψ is the right adjunct. The necessary and sufficient condition for two functors to be adjoint is that between the two-variable functors Hom(Φ-,-):$MSC_1\to$ Set and Hom(-,Ψ-):$MSC_2\to$Set, there must be a bijective natural transformation, i.e. we have the natural isomorphism Hom(Φ-,-)$\cong$Hom(-,Ψ-), where we have denoted with "-" the place of a variable. This condition generalizes the condition from the Galois connection. In other words, in the MSC_1 and MSC_2 categories, for each pair of objects $\mathfrak{I}_p\in\mathfrak{I}$ and $\mathfrak{L}_q\in\mathfrak{L}$, there is a behavioral transformation from $\Phi(\mathfrak{I}_p)$ to $\mathfrak{L}_q$ in the MSC_2 category if and only if there is a transformation from $\mathfrak{I}_p$ to $\Psi(\mathfrak{L}_q)$ in the MSC_1 category. But this condition is exactly the necessary and sufficient condition to be able to carry out a co-simulation without deadlock on all the simulation traces defined by the MSC_1 and MSC_2 categories.

Therefore, the necessary and sufficient condition for the functors Φ and Ψ to form an adjunction is for the natural isomorphism Hom(Φ-,-)≅Hom(-,Ψ-) to exist. From here it follows that in order to define the functors Φ and Ψ so that they form an adjunction $\Phi \dashv \Psi$, it is enough to define a one-to-one relationship between the traces from the categories of simulation models MSC_1 and MSC_2. This correspondence between the simulation traces is reduced to a bijective natural transformation defined on components $f{:}Hom(\Phi\mathfrak{J}_p,\mathfrak{L}_q)\rightarrow Hom(\mathfrak{J}_p,\Psi\mathfrak{L}_q)$, between the behavioral rules of the two behavioral models. Thus if $\tau_2 \in Hom(\Phi\mathfrak{J}_p,\mathfrak{L}_q)$ then $\Psi(\tau_2)=f^{-1}(\tau_2)$, and if $\tau_1 \in Hom(\mathfrak{J}_p,\Psi\mathfrak{L}_q)$, then $\Phi(\tau_1)=f(\tau_1)$. Also, if in the state $\mathfrak{J}_p\in\mathfrak{J}$ of the MSC_1 model, we have a request to the MSC_2 model, which is in the state $\mathfrak{L}_r$, and we need an answer in the $\mathfrak{J}_r\in\mathfrak{J}$ state, from the MSC_2 model, in the $\mathfrak{L}_q\in\mathfrak{L}$ state, then we define $\Phi\mathfrak{J}_p= \mathfrak{L}_r$ and $\Psi\mathfrak{L}_q = \mathfrak{J}_t$.

In the context of two adjunct functors we can define two special natural transformations called unit and counit. [10, 12]. For the two adjunct functors $\Phi \dashv \Psi$, there is a natural transformation $\eta{:}id_1\rightarrow\Phi\circ\Psi$, where id_1, is the identity functor from the MSC_1 category, so that for any object $\mathfrak{J}_k\in\mathfrak{J}$ and $\mathfrak{L}_l\in\mathfrak{L}$ and any arrow $\tau_1^{kt}{:}\mathfrak{J}_k\rightarrow\Psi(\mathfrak{L}_l)=\mathfrak{J}_t$, there is a unique arrow $\tau_2^{tl}{:}\Phi(\mathfrak{J}_k)=\mathfrak{L}_t\rightarrow\mathfrak{L}_l$ so that the diagram in Fig. 3 commutes [10, 12]. The natural transformation η is called unit adjunction.

Also, the adjunction property of the functors Φ and Ψ assumes the existence of a dual natural transformation $\varepsilon{:}\Psi\circ\Phi\rightarrow id_2$, where id_2 is the identity functor from the category MSC_2, so that for any arrow $\tau_2^{tl}{:}\Phi(\mathfrak{J}_k)=\mathfrak{L}_t\rightarrow\mathfrak{L}_l$, there is a unique arrow $\tau_1^{kt}{:}\mathfrak{J}_k\rightarrow\Psi(\mathfrak{L}_l)= \mathfrak{L}_t$, so that the diagram in Fig. 4 commutes. The natural transformation ε is called counit adjunction.

The adjunction of two functors $\Phi \dashv \Psi$ is unambiguously specified by the tuple (Φ,Ψ,η,ε), where η is the adjunction unit and ε is the adjunction counit.

The adjunction unit $\eta{:}id_1\rightarrow\Phi\circ\Psi$ is also called the insertion of generators and has the role of transforming each object $\mathfrak{J}_k\in\mathfrak{J}$ into the format of an object $\Psi(\Phi(\mathfrak{J}_k))$. This function is executed in the MSC_1 model and can be calculated on components starting from the adjunction condition which says that there is a natural isomorphism $f{:}Hom(\Phi\mathfrak{J}_p,\mathfrak{L}_q)\rightarrow Hom(\mathfrak{J}_p,\Psi\mathfrak{L}_q)$. If in this natural isomorphism we make $\mathfrak{L}_q=\Phi\mathfrak{J}_p$, then we have the natural isomorphism $f{:}Hom(\Phi\mathfrak{J}_p,\Phi\mathfrak{J}_p)\rightarrow Hom(\mathfrak{J}_p,\Psi\Phi\mathfrak{J}_p)$ and therefore we can calculate η on the $\mathfrak{J}_p$ component, according to the formula: $\eta_{\mathfrak{J}_p}=f(id_{\Phi\mathfrak{J}_p}$ where $id_{\Phi\mathfrak{J}_p}{:}\Phi(\mathfrak{J}_p)\rightarrow\Phi(\mathfrak{J}_p)$ is the identity functor in MSC_2.

Similarly, we have the natural isomorphism $f{:}Hom(\Phi\Psi\mathfrak{L}_q,\mathfrak{L}_q)\rightarrow Hom(\Psi\mathfrak{L}_q,\Psi\mathfrak{L}_q)$ and therefore we can also calculate the adjunction counit $\varepsilon{:}\Psi\circ\Phi\rightarrow id_2$ on components according to the formula: $\varepsilon_{\mathfrak{L}_k} = f^{-1}(id_{\Psi\mathfrak{L}_k})$ where $id_{\Psi\mathfrak{L}_k}{:}\Psi(\mathfrak{L}_k)\rightarrow\Psi(\mathfrak{L}_k)$ is the identity functor in MSC_1. This function is executed in the MSC_2 model and represents, in our case, the process of generating the response of the MSC_2 model to the request of the MSC_1 model.

The adjoint of two functors defined by the tuple (Φ,Ψ,η,ε), where Φ and Ψ are adjoint functors, η is the adjoint unit and ε is the adjoint counit, determines a monad [11]. An endofunctor $T{:}MSC_1\rightarrow MSC_1$, together with two natural transformations $\eta{:}id \rightarrow T$, called "return", and $\mu{:}T^2\rightarrow T$, called "join", which make the diagrams in Fig. 5 and Fig. 6 commutative is a monad in category MSC_1, which is specified by a tuple (T,η,μ).

Based on the adjunction (Φ,Ψ,η,ε), we can calculate the monad components as follows: the endofunctor $T=\Phi\circ\Psi$, η is the adjunct unit, and $\mu=\Psi\varepsilon\Phi$ [11]. Monads are frequently used in functional languages that offer in this way facilitates for inserting imperative code into the functional code. Therefore, in a similar way, mechanisms for specifying monads can be introduced in domain specific modeling languages [6].

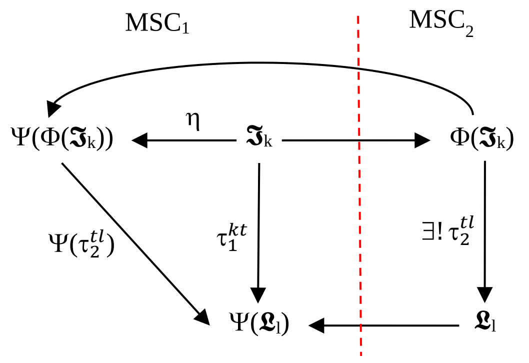
$$\begin{tikzcd}
 & MSC_1 & MSC_2 \\
\Psi(\Phi(\mathfrak{J}_k)) \arrow[dr, "\Psi(\tau_2^{tl})"'] & \mathfrak{J}_k \arrow[l, "\eta"'] \arrow[r] \arrow[d, "\tau_1^{kt}"] & \Phi(\mathfrak{J}_k) \arrow[ll, bend right] \arrow[d, "\exists!\,\tau_2^{tl}"] \\
 & \Psi(\mathfrak{L}_l) & \mathfrak{L}_l \arrow[l]
\end{tikzcd}$$

Fig. 3. Definition of Unit Adjunction.

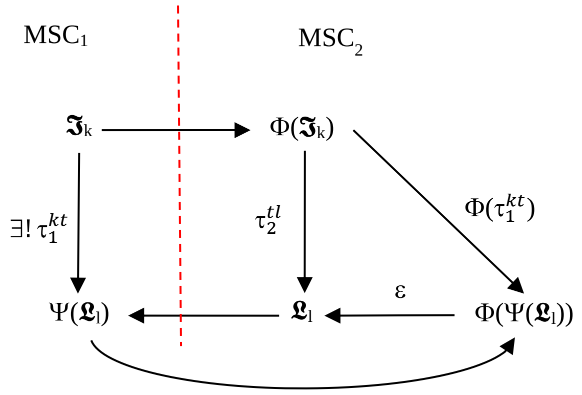
$$\begin{tikzcd}
MSC_1 & MSC_2 & \\
\mathfrak{J}_k \arrow[r] \arrow[d, "\exists!\,\tau_1^{kt}"'] & \Phi(\mathfrak{J}_k) \arrow[d, "\tau_2^{tl}"] \arrow[dr, "\Phi(\tau_1^{kt})"] & \\
\Psi(\mathfrak{L}_l) \arrow[rr, bend right] & \mathfrak{L}_l \arrow[l] & \Phi(\Psi(\mathfrak{L}_l)) \arrow[l, "\varepsilon"']
\end{tikzcd}$$

Fig. 4. Definition of Counit Adjunction.

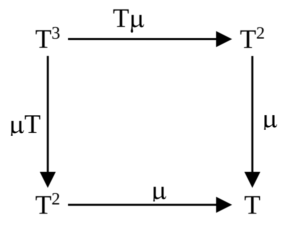
$$\begin{tikzcd}
T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"] & T
\end{tikzcd}$$

Fig. 5. First Constraint.

$$\begin{tikzcd}
T \arrow[dr, "id"'] & T^2 \arrow[l, "\eta T"'] \arrow[r, "T\eta"] \arrow[d, "\mu"] & T \arrow[dl, "id"] \\
 & T &
\end{tikzcd}$$

Fig. 6. Second Constraint.

Most of the time, the co-simulation control is realized by an orchestrator that coordinates the system components according to a co-simulation scenario [3,25]. In this case, the two models must be encapsulated in software units that implement a standard interface [2]. The interaction between two CPPS models can be specified by defining shared variables between the two models or by passing data to the orchestrator. In the case of our approach, the adjunction unit η is executed in the MSC_1 model and has the role of preparing the data, in the appropriate format, to be transmitted to the orchestrator. The μ application is executed in the MSC_1 model, and specifies the operations to be executed by this model. Connecting the models through monads makes it possible to analyze the composite system resulting from co-simulation.

V. Original Contributions and Conclusions

The most important finding, in our proposal, is simplicity and conceptual clarity. Thus, the static dimension of a model is the image of a categorical sketch through a functor. The behavioral dimension of a model is specified by a set of functors and a set of behavioral rules. The simulation space of a model is defined by a category. The co-simulation space of two models is specified by a monad induced by two adjunct functors. All the mechanisms involved in these definitions are generic and can be implemented at the metamodel level. Mechanisms for specifying monads can be included in the modeling language at the metamodel level, as happens in functional languages that offer such mechanisms especially to allow imperative specifications. Most of the times the interaction between two CPPS models can be specified by defining shared variables. Connecting the models through monads makes it possible to analyze the composite system resulting from co-simulation. To our knowledge, co-simulation through monads, which is the main objective of this work, has not been addressed until now.

References

[1] Henderik A. Proper and Giancarlo Guizzardi, "On Domain Conceptualization" Advances in Enterprise Engineering XIV, EEWC 2020, Bozen-Bolzano, Italy, September 28, October 19, and November 9–10, 2020 ; Springer 2021.

[2] Functional Mock-up Interface for Model Exchange and Co-Simulation,Document version: 2.0.1 October 2nd 2019, https://fmi-standard.org/.

[3] INTO-CPS Tool Chain User Manual, Deliverable Number: D4.3a Version: 1.0 Date: December, 2017 Public Document, http://into-cps.au.dk.

[4] D. Karagiannis, H.C. Mayr, J. Mylopoulos, "Domain-Specific Conceptual Modeling Concepts, Methods and Tools" Springer International Publishing Switzerland (2016).

[5] Dominik Bork, Dimitris Karagiannis, Benedikt Pittl, "A survey of modeling language specification techniques", Information Systems 87 (2020) 101425, journal homepage: www.elsevier.com/locate/is.

[6] M. Fowler, R. Parsons, "Domain Specific Languages", 1st ed. Addison-Wesley Longman, Amsterdam, 2010.

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

[8] D.C. Crăciunean, "Categorical Grammars for Processes Modeling", International Journal of Advanced Computer Science and Applications(IJACSA), 10(1), (2019).

[9] D.C. Crăciunean, D. Karagiannis, "A categorical model of process co-simulation", Journal of Advanced Computer Science and Applications(IJACSA), 10(2), (2019).

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

[11] Michael Barr Charles Wells. "Toposes, Triples and Theories" November 2002.

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

[13] Zinovy Diskin, Tom Maibaum- "Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond", ACCAT 2012.

[14] Diskin Z., König H., Lawford M., „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.

[15] Uwe Wolter, Zinovy Diskin, "The Next Hundred Diagrammatic Specification Techniques, A Gentle Introduction to Generalized Sketches", 02 September 2015 : https://www.researchgate.net/publication/253963677,

[16] D. Plump, "Computing by graph transformation: 2018/19", Department of Computer Science, University of York, UK, Lecture Slides, 2019.

[17] G. Campbell, B. Courtehoute and D. Plump, "Linear-time graph algorithms in GP2", Department of Computer Science, University of York, UK, Submitted for publication, 2019. [Online]. Available: https://cdn.gjcampbell.co.uk/2019/Linear-Time-GP2-Preprint.pdf.

[18] D. Plump, "Checking graph-transformation systems for confluence", ECEASST, vol. 26, 2010. DOI: 10.14279/tuj.eceasst.26.367.

[19] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, Frank Hermann, "Graph and Model Transformation General Framework and Applications", Springer-Verlag Berlin Heidelberg 2015.

[20] R. Milner, "The Space and Motion of Communicating Agents", Cambridge University Press, (2009).

[21] C. Gomes, C. Thule, D. Broman, P.G. Larsen, H. Vangheluwe, "Co-simulation: State of the art", ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January (2016).

[22] Claudio Gomes, Casper Thule, Levi Lucio, Hans Vangheluwe, and Peter Gorm Larsen, "Generation of Co-simulation Algorithms Subject to Simulator Contracts", https://sites.google.com/view/cosimcps19.

[23] David I. Spivak, "Category Theory for the Sciences", The MIT Press Cambridge, Massachusetts London, England, 2014 Massachusetts Institute of Technology.

[24] Brendan Fong and David I. Spivak, "Seven Sketches in Compositionality" An Invitation to Applied Category Theory; 2018.

[25] Bottaccioli, Lorenzo; Estebsari, Abouzar; Pons, Enrico; Bompard, Ettore; Macii, Enrico; Patti, Edoardo; Acquaviva, Andrea, "A Flexible Distributed Infrastructure for Real-Time Co-Simulations in Smart Grids" in: IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, vol. 13 n. 6, pp. 3265-3274. - ISSN 1551-3203, (2017).