Categorical Grammars for Processes Modeling

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

***Abstract*—The diversity and heterogeneity of real-world systems makes it impossible to naturally model them only with existing modeling languages. For this reason, models are often constructed using domain specific modeling languages as metamodels, which must themselves be specified by meta-metamodels. In this paper we present a new approach, based on the category theory, to specify metamodels. A grammar for modeling processes (PN, CSP, EPC, etc.) syntactically defines processes and then presents a set of reaction rules that model the behavior of the system. We will see that the categorical sketch is sufficiently expressive to be able to support the constructions needed to visually define the syntax of a graphical modeling language. The category theory also provides appropriate structures to model the behavioral rules of a real system.**

***Keywords*—Process modeling; metamodel; modeling grammars; categorical grammars; category theory; categorical sketch**

I. Introduction

In the theory of systems, we can distinguish between the structure of a system and the behavior of the system. The structure is the internal organization of a system. The operational aspect of the structure is given by a set of objects that are invariant to transformations.

An important method of mathematical modeling of the behavior of a dynamic system is provided by the process concept [9]. We understand a process as a behavioral model of a dynamic system at a certain level of abstraction. The behavior of the system generally consists of processes and data.

Processes are control mechanisms for data manipulation. Processes are dynamic and active, data is static and passive. The data of a process is generally expressed by ontologies that allow for inference, and processes through directed graphs whose nodes are states and arcs are actions.

While a sequential system performs a single step at a time and can therefore be characterized by a single current state, the different components of a concurrent system may be in different local states at a time, which together make up the global state of the system at a time [12]. Furthermore, intermediate states are as important as the initial state and the final state, as they determine the behavior of larger systems that may include the considered system as a component.

The behavior of the system is given by several processes that are executed simultaneously (parallel and distributed), where these processes exchange data to influence each other's evolution.

Due to the different component execution speeds, the way the components interact with each other, and the programming policies adopted, the behavior of these competing systems may present interesting situations such as non-determinism in the end result or in the actual calculation. Consequently, it is not appropriate to describe the behavior of these systems by a function from inputs to outputs, as in the classical theory of systems [12,13].

A process is a sequence of steps that define behavior. There are several approaches to the notion of step, which leads to as many different types of behavior. Most often, the process models visually describe how real systems work [2,3,4,5].

The grammar of a visual language defines the syntax generation rules and the semantic interpretation rules of graphical elements in a process model, as well as the rules of composition of atomic components to model the behavior of the real system. It is important that the rules of syntactic construction be such that any model generated on their basis allows for a detailed syntactic analysis, that is, to allow the determination of the sequence of syntactic rules that generated the model. This succession of syntactic rules allows semantic interpretation of the model. [4,11,14].

Different modeling grammars tend to emphasize different aspects of processes, i.e. a Petri Net model of a real problem looks different from an EPC model of the same real problem. Consequently, the choice of modeling grammar is an essential decision when the modeling activity begins [11,14].

Generally, building a model begins with an informal model, used for discussion and documenting, and ends with an executable model useful for analyzing, simulating, or actually executing the process [11,14].

Informal models are easy to understand but suffer from ambiguity, while executable models are too detailed to be easy to understand by all the parties involved in building the model.

This conflict between the informal and the executable model largely reflects a certain incompatibility between the metamodel and the modeled object, and therefore is mainly due to the insufficient alignment between the metamodel and reality.

The diversity and heterogeneity of real-world systems makes it impossible to naturally model them only with existing modeling languages [4,14].

The metamodel requires an abstract version of reality, focuses on the common behavior of the real systems in question, and therefore the metamodel cannot cover satisfactorily only a small percentage of the actual cases that its authors consider to be representative [10].

Of course, these drawbacks can be solved by successive upgrades of existing metamodels with new structures, concepts and algorithms, but these additions often exceed the initial logic of the metamodel. Therefore this method of solving the drawbacks leads to difficult to master graphical languages.

On the other hand, a modeling grammar can be specific to certain aspects of processes, such as flow of activities, allocation of resources, communication between processes, etc. Obviously, the solution to this problem, if costs are acceptable, is given by the languages specific to each modeling domain that may contain elements representative of the concepts involved in a particular modeling domain.

This approach requires powerful and flexible metamodeling tools to support the specification and generation of domain specific modeling languages with acceptable costs. The specification of such a metamodel should contain enough information to allow the automatic generation of a tool to verify and build models subject to the syntax of the described formalism.

In this paper we will show that the sketches from the category theory offer a language with a well-defined syntax and semantic to describe mathematical objects, that can rigorously represent the syntax of domain-specific modeling languages.

We will see that categorical sketches are mathematical objects with well-defined syntax and semantics that represent meta-metamodels capable of capturing the basic elements that can be used to design a metamodeling formalism. In this context, a metamodel is represented by a mathematical object, a sketch, and a model is a functor that is also a mathematical object.

The fact that the sketch is a graphical specification makes the metamodel specification process intuitive, accessible and reduces the time to develop a modeling tool.

In section 2 we present the theoretical foundations and notations in the category theory. Section 3 presents the use of the categorical sketch of the process model concept. Section 4 defines the metamodel as a functor, and section 5 completes the model with the execution and simulation part.

II. Theoretical Foundations and Notes

Definition 1. [1,6,7,9] A category $\mathcal{C}$ consist of a set of objects, a set of arrows between these objects, and a partial operation of arrows composition. We will denote the category objects with uppercase letters A, B, ..., the set of all objects we will denote with ob($\mathcal{C}$), the set of arrows between two objects X and Y with $\mathcal{C}$(X,Y) and the partial operation of arrows composition with $\circ$. The set of arrows of a category $\mathcal{C}$ along with the arrows composition operation form a monoid structure, i.e. it is associative: for all arrows $f\in\mathcal{C}(X,Y)$, $g\in\mathcal{C}(Y,Z)$ and $h\in\mathcal{C}(Z,W)\Rightarrow(h\circ g)\circ f=h\circ(g\circ f)\in\mathcal{C}(X,W)$, and for each object X in ob($\mathcal{C}$) there is an identity arrow $id_X:X\rightarrow X$ with the property $id_X\circ f=f$, $g\circ id_X=g$ where $X,Y,U\in ob(\mathcal{C})$, $f\in\mathcal{C}(Y,X)$ and $g\in\mathcal{C}(X,U)$.

Definition 2. [1,7,8,9] A functor ϕ is an application between two categories $\mathcal{C}$ and $\mathcal{D}$ that maps the objects of category $\mathcal{C}$ into objects of category $\mathcal{D}$ and the arrows of category $\mathcal{C}$ in arrows of category $\mathcal{D}$ with the preservation of the structure, i.e.: $\phi_{A,B}:\mathcal{C}(A,B)\rightarrow\mathcal{D}(\phi(A),\phi(B))$ for all objects $A,B\in ob(\mathcal{C})$ and $\phi(1_A)=1_{\phi(A)}$, $\phi(fg)=\phi f\phi g$ where $X,Y,Z\in ob(\mathcal{C})$, $g\in\mathcal{C}(X,Y)$, $f\in\mathcal{C}(Y,Z)$.

If we consider each category an object and each functor an arrow between these objects we get a category that is usually denoted with Cat.

Definition 3. [1,7,8,9] A natural transformation is an application between two functors ϕ and ψ which have the same domain $\mathcal{C}$ and the same codomain $\mathcal{D}$ consisting of a family of arcs $\tau_A:\phi A\rightarrow\psi A$ ($A\in\mathcal{C}$) such that for each arrow $f:A\rightarrow B$ in $\mathcal{C}$, the naturality condition is respected $(\psi f)\circ\tau_A=\tau_B\circ(\phi f)$.

A small category could be defined as a graph $\mathcal{G}$ to which a structure is added, i.e. an arc composition operation and an identity arc for each node. In this way, any graph $\mathcal{G}$ generates a category called the free category generated by the graph $\mathcal{G}$. This is very important for visual models because they are generally graphs generated on the basis of the syntactic rules imposed by the corresponding grammars. Therefore, any process model generates a free category.

The operation of generating the free categories from graphs also involves the extension of graph homomorphisms to the corresponding functors between the free categories generated by them. Based on this observation, to simplify the exposure, we will use the functor designation for graphs as well. However, we must note that if there is always a functor between two categories (at least one constant functor), there is not always a homomorphism between two graphs.

Definition 4. [1,7,8] A diagram is a functor D defined on a graph $\mathcal{G}$ with values in another graph $\mathcal{P}$ or with values in a category $\mathcal{C}$. The domain of D is called shape graph of diagram D.

Definition 5. [1,7,8] A commutative cone in category $\mathcal{C}$ with the vertex $C\in\mathcal{C}$ and the base a diagram $D:\mathcal{G}\rightarrow\mathcal{C}$ is a natural transformation $p:\Delta_C\rightarrow D$ where Δ_C is the constant diagram $\Delta_C:\mathcal{G}\rightarrow\mathcal{C}$.

A morphism between two cones p and p' is an arrow $f:C\rightarrow C'$ with the property that for any node a of the graph $\mathcal{G}$ we have $p_a= p'_a$ f. The set of cones together with these morphisms form the cone category generated by diagram D.

Definition 6. [1,7,8] The limit of a diagram $D:\mathcal{P}\rightarrow\mathcal{C}$ is a terminal object in the cone category generated by diagram D.

Definition 7. [1,7,8] A commutative cocone in the category $\mathcal{C}$ with the vertex $C\in\mathcal{C}$ and the base a diagram $D:\mathcal{G}\rightarrow\mathcal{C}$ is a natural transformation $p:D\rightarrow\Delta_C$ where Δ_C is the constant diagram $\Delta_C:\rightarrow\mathcal{C}$.

Definition 8. [1,7,8] A morphism between two cocones p and p' is an arrow f:C→C′ with the property that for any node a of the graph $\mathcal{G}$ we have $p'_a=fp_a$. The set of cocones along with these morphisms form the cocone category generated by diagram D.

Definition 9. [1,7,8] The colimit of a diagram D:$\mathcal{G}\rightarrow\mathcal{C}$ is an initial object in the cocone category generated by diagram D.

Definition 10. [1,7,8] A categorical sketch $\mathcal{S}$ is a tuple ($\mathcal{G}$, $\mathcal{D}$, $\mathcal{L}$, $\mathcal{K}$) where $\mathcal{G}$ is a graph, $\mathcal{D}$ is a set of diagrams, $\mathcal{L}$ is a set of cones and $\mathcal{K}$ a set of cocones.

Definition 11. [1,7,8] A model generated by sketch $\mathcal{S}$=($\mathcal{G}$, $\mathcal{D}$, $\mathcal{L}$, $\mathcal{K}$) is a functor M:$\mathcal{G}$→Set that maps the diagrams D to commutative diagrams, the cones $\mathcal{L}$ to cone limits and the cocones $\mathcal{K}$ to cocone colimits in Set.

III. Categorical Sketch of the Process Model

Essentially, a visual model of a process defines first the syntax of the process that represents the virtual and physical entities of the model and then the semantics of the process represented by a set of reaction rules that represent the behavior of these entities [10].

The syntax of a process can be represented by graphs that have as nodes specific concepts and as arcs the interdependencies between these concepts. Often the syntax of the model can be represented by a single graph. When models of a real systems imply a space concept, an additional graph is used that has the same nodes as the first, thus reaching the notion of bigraph [10]. In this paper we will only deal with processes that can be represented by a graph.

In the Set category, a graph is defined by two sets X, Γ and two parallel functions σ, θ defined as in Fig. 1. To specify the syntax of a graphical metamodel we will use a categorical sketch, which in turn is represented in a graphical language [10].

Sketches are not designed as a notation, but as a mathematical structure that incorporates an exact formal syntax and semantics. We will use the same notations for the arcs of the graph of the sketch and the functions from Set, and the nodes from the graph of the sketch we will denote with lowercase letters and the objects from Set we will denote with upercase letters.

We could therefore consider the starting point in defining a sketch corresponding to the concept of process a graph with two nodes x, γ and two parallel arcs σ and θ. However, this sketch is too general and does not in any way account for the specifics and restrictions of each metamodel.

Therefore, we will need to introduce a series of helper objects and functions in the Set category to impose the constraints specific to each metamodel. These helper objects will be reflected in the sketch components (the graph of the sketch, commutative diagrams, cones and cocones).

Below we will present some of these possible constructions and we will also present a relevant example.

$$\begin{tikzcd} \boxed{\Gamma} \arrow[r, shift left, "\sigma"] \arrow[r, shift right, "\theta"'] & \boxed{X} \end{tikzcd}$$

Fig. 1. Graph Sketch.

$$\begin{tikzcd} & \boxed{\Gamma} \arrow[dl, "\sigma"'] \arrow[d, "\mu"] \arrow[dr, "\theta"] & \\ \boxed{X} & \boxed{X\times X} \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & \boxed{X} \end{tikzcd}$$

Fig. 2. Commutative Diagram.

$$\begin{tikzcd} \boxed{\Gamma} \arrow[r, "\iota_\gamma"] \arrow[d, "\iota_\gamma"'] & \boxed{\Gamma} \arrow[d, "\mu"] \\ \boxed{\Gamma} \arrow[r, "\mu"] & \boxed{X\times X} \end{tikzcd}$$

Fig. 3. Pullback Diagram.

A. *It's a Simple Graph, not a Multigraph*

A simple graph is a graph with the property that for any pair (a, b) of vertices there is no more than one arrow with source a and target b. In order to impose this condition in the Set category, we need the Cartesian product X×X and the function μ defined as shown in Fig. 2. The functions π_1 and π_2 are the two projections.

To get a sketch that specifies only simple graphs, we add an object x×x and the discreet cone needed to convert this object into a formal product. Then we have a single arc between any two vertices if and only if μ:Γ→X×X is a monomorphism. But the function μ is a monomorphism in Set if and only if the pullback of μ with μ is equal to Γ, i.e. if and only if the diagram in Fig. 3 is a pullback diagram. The effect of this is to make the monic arrow μ become a pair <s,t> in a model so that there are no two arrows to have the same pair source, destination.

B. *The Graph must be Connected*

In order to constrain the graph corresponding to a model to be connected, we will define a function ν:X→U that associates to each object in X the connected component to which it belongs. So U is the set of connected graph components. But this ν is a coequalizer for the functions σ and θ.

A coequalizer ν:X→U must satisfy the equality ν∘σ = ν∘θ.

The pair (σ, θ) determines a relation ρ*⊆X×X which is obtained by the reflexive, symmetrical and transitive closure of the relation ρ={(σ(t), θ(t))|t∈X}.

Obviously we have ν∘σ = ν∘θ if and only if $\nu(t_1)=\nu(t_2)$ for all $(t_1, t_2)\in\rho^*$.

$$\begin{tikzcd} \Gamma \arrow[r, shift left, "\sigma"] \arrow[r, shift right, "\theta"'] & X \arrow[r, "\nu"] & U \end{tikzcd}$$

Fig. 4. Coequalizer.

We will define $U=X/_{\rho*}$, i.e. the set of equivalence classes determined by ρ and ν:X→U is the function that associates to an element from X its equivalence class, i.e. the connected component to which it belongs. But the function μ is a coequalizer of σ and θ (Fig. 4).

Therefore, U is the colimit of a diagram with two nodes γ, x and two arcs σ, θ and it will supply the connected components of our graph. But we want the graph to be connected and therefore to have only one connected component [7,8].

For this we will put the condition that U is the vertex of a cone with an empty base. Thus, U will become an object in our model from Set, i.e. a set with a single element, which guarantees that our graph will be connected.

Another method to specify that a graph is connected is that the diagram from Fig. 5 has to be a pushout diagram. That is, the pushout of σ with θ is ω (a terminal object in Set).

1) The types of objects determine a partition on the set of the graphs vertices: X=X1∪X1 and X1∩X2=∅ . In the model sketch, the disjoint union X is the colimit of a discrete diagram (cocone).

2) The types of arcs determine a partition on the set of arcs of the graph: $\Gamma=\Gamma_1\cup \Gamma_2$ and $\Gamma_1\cap \Gamma_2=\emptyset$. In the sketch of the model the disjoint union Γ is the colimit of a discrete diagram (cocone).

3) The maximum or minimum number of arcs coming out of a vertex or entering a restricted vertex. We will denote by

$\Gamma x=\{ y|(x,y)\in \Gamma\}$ and with $\Gamma^{-1}x=\{ y|(y,x)\in \Gamma\}$.

The sequential routing involves the activation of an activity in a process, always, after completing another activity in the same process.

The sequence model is used to model consecutive steps in a process, whatever the case, and is supported by all available process management systems. In most cases, two activities that execute sequentially depend on each other in the sense that the second one uses the result of the first one. Typical implementation involves tying two activities with an unconditional control arrow [12,13,14].

From a syntactical point of view, the sketch corresponding to the metamodel will require to put a constrain that the first activity be the source of a single arrow and the second one to be the target of a single arrow.

That is, if we have two sets of activities X_1 and X_2 so that, always, an activity of X_1 is followed by a single activity from X_2 and only that, i.e. it complies with the sequential routing, then in the metamodel sketch we will have a diagram of the form Fig. 6 with the property that σ and θ are monomorphism.

$$\begin{tikzcd} \Gamma \arrow[r, "\sigma"] \arrow[d, "\theta"'] & X \arrow[d] \\ X \arrow[r] & \omega \end{tikzcd}$$

Fig. 5. Pushout Diagram.

$$\begin{tikzcd} & X_1 \\ \Gamma_{1,2} \arrow[ur, "\sigma"] \arrow[dr, "\theta"'] & \\ & X_2 \end{tikzcd}$$

Fig. 6. Sequential Routing.

$$\begin{tikzcd} \Gamma_{1,2} \arrow[r, "\iota_A"] \arrow[d, "\iota_A"'] & \Gamma_{1,2} \arrow[d, "\sigma"] \\ \Gamma_{1,2} \arrow[r, "\sigma"'] & X_1 \end{tikzcd}$$

Fig. 7. Pullback Diagram.

$$\begin{tikzcd} \Gamma_{1,2} \arrow[r, "\iota_A"] \arrow[d, "\iota_A"'] & \Gamma_{1,2} \arrow[d, "\theta"] \\ \Gamma_{1,2} \arrow[r, "\theta"'] & X_1 \end{tikzcd}$$

Fig. 8. Pullback Diagram.

In the metamodel sketch we will have to impose the condition that θ and σ are monomorphisms, i.e. an object in X_1 can be followed by a single object and an object in X_2 can be preceded by a single object.

But, θ and σ are monomorphisms if and only if the pullback of θ with θ is $\Gamma_{1,2}$ and the pullback of σ with σ is $\Gamma_{1,2}$, i.e. the diagrams in Fig. 7 and Fig. 8 have to be pullback diagrams.

The condition that a concept from the set X_1 can be followed by any number of concepts from X_2 and a concept from X_2 can be preceded by a single concept from X_1 is that θ be a monomorphism, i.e. the pullback of θ with θ is $\Gamma_{1,2}$. (Fig. 8).

The condition that a concept from the set X_1 can be followed by a single concept from X_2, and a concept from X_2 can be preceded by any number of concepts from X_1 is that σ be a monomorphism, i.e. the pullback of σ with σ is $\Gamma_{1,2}$. (Fig. 7).

If we do not put constraints on a subgraph like the one in Fig. 6 from the graph of the sketch, then an object from X_1 can be followed by any number of objects from X_2, and an object from X_2 can be preceded by any number of objects from X_1.

Example 1. Medical Laser Manufacturing Systems (MLMS).

At a medical lasers company, a software tool is required for modeling and simulating a manufacturing cell that assembles multiple devices simultaneously. The assembly process also contains common operations on such devices.

A cell can have a set of input buffers X_I (entry station), a set of output buffers X_O (exit station), a set of workstations, w_0, w_1, …, w_{n-1}, a set of test stations and a set of buffers to collect faulty components. These workstations are loaded and unloaded by a set of specific conveyors.

The manufacture of each device is made in accordance with its process plan. There are several types of devices with specified process plans.

The primary components of a device reach an entry station. Once the primary components reach this point, they are inserted into the assembly system when possible. They will be transported and assembled in workstations, in accordance with the process plan and then leave the system via an exit station or through a collection station for faulty components.

Each workstation w_i has an input buffer B_i and an output buffer B_o that have limited capacities. A workstation works asynchronously if it has raw material in the input buffer and enough space in the output buffer. If one of these conditions is not met, the station stops and starts automatically when the conditions are met. The assembly operation has a certain duration.

Each test station X_t has an input buffer B_i and an output buffer B_o with limited capacities. A test station works asynchronously if it has raw material in the input buffer and enough space in the output buffer. If one of these conditions is not met, the station stops and starts automatically when the conditions are met. The test operation has a certain duration.

Each conveyor Γ has a limited transport capacity and can carry several types of components in specified quantities. A conveyor works asynchronously if it has sufficient components in the output buffer of the source workstation and also has enough space in the input buffer of the target workstation. If one of these conditions is not met the conveyor stops and starts automatically when the conditions are met. The transport operation has a certain duration.

Each input buffer X_i has the ability to store several types of components in limited quantities, and we assume it is continuously supplied from the outside of the model. Each output buffer X_o has the ability to store, in limited quantities, more types of finished products and we assume it is emptied from the outside of the model. Also, each buffer for collecting faulty components X_d has the ability to store, in limited quantities, a number of defective components and we assume it is emptied from the outside of the model.

The purpose of the model is to evaluate the performance of the manufacturing cell or to investigate different programming policies in order to optimize the manufacturing process. For this purpose, information was included in the model, such as the duration of operations, the stop time of the actions in order to locate the process delay points. The model also allows optimizing the size of the buffers in order to eliminate stagnation due to downstream or upstream defects.

As a result of the analysis, we find that in order to graphically specify such processes we need 6 types of concepts, namely: input buffers with primary components, output buffers with finished products, faulty components collection buffers, assembly stations, test stations and conveyors.

The workflow also includes the following routing rules:

At the beginning of the manufacturing process, the primary components will pass through a test station.

After each assembly station, a test station will be required. Components assembled in a workstation will always go to the same test station.

In the test station the components will be sorted into accepted and defective. Accepted components will follow the assembly flow and the defective components will be transported to a collection buffer for faulty components.

We will define an MLMS as a graph with a set of syntactic restrictions. The mechanisms used to introduce the syntactic constraints of the models are those from the sketch definition, i.e. commutative diagrams, limits and colimits.

Definition 12. A MLMS model is a directed graph

$\mathcal{G} = (X, \Gamma, \sigma, \theta)$ where

X is a set of objects (concepts in our model) that represent the nodes of the graph.

Γ is a set of arcs (conveyors in our model).

And which satisfies the following properties:

1) $\mathcal{G}$ is a connected graph
2) There is only one arc between any two nodes.
3) On the set of nodes X we have a partition: $X=X_i \sqcup X_o \sqcup X_d \sqcup X_w \sqcup X_t$

Where
Xi is a set of input buffers for the primary components;
Xo is a set of output buffers for finished products;
Xd is a set of collection buffers for faulty components;
Xw is a set of assembly stations;
Xt is a set of testing stations.

4) σ and θ are functions $\sigma,\theta:\Gamma\rightarrow X$ which assigns to each arc $r\in\Gamma$ the source and target objects $\sigma(r), \theta(r)\in X$.
$\Gamma\subseteq(X_i\times X_t)\cup(X_t\times X_w)\cup(X_w\times X_t)\cup(X_t\times X_o)\cup(X_t\times X_d)$
$\Gamma=\Gamma_{it}\sqcup\Gamma_{tw}\sqcup\Gamma_{wt}\sqcup\Gamma_{to}\cup\Gamma_{td}$
5) $\Gamma x|=1$ for any $x\in X_w$, i.e. the components assembled in a workstation will all go to the same test station.

As we can see the syntactic definition of an MLMS, introduces a series of partitions on the set of nodes, subpartitions on the set of events, connectors, and arcs. Also, the definition includes connection constraints and number of arcs between different types of nodes.

So far we have built the sketch of the graph into several components that can be aggregated in a related graph as in Fig. 9. As we notice, we have introduced all the nodes and arcs in the graph of the sketch so that we can define the components $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$ that introduce the constraints specific to our metamodel. Any model of the resulting sketch will comply with these constraints.

Graph $\mathcal{G}$ has 14 nodes and 27 arrows. These will be interpreted in a model as follows: (1) x - all object X in a MLMS model, (2) X_i – is a set of input buffers for the primary components, (3) X_o is a set of output buffers for finished products, (4) X_d is a set of collection buffers for faulty components, (5) X_w is a set of assembly stations, (6) X_t is a set of testing stations (7) x×x - the Cartesian product of the set X with X, (8) ω represents a terminal object in Set, (9) γ - represents all relations Γ between the objects of the model, (10) γ_{it} - represents the subset of relations Γ_{it} that links X_i objects with X_t objects, (11) γ_{tw} - represents the subset of relations Γ_{tw} that links X_t objects with X_w objects, (12) γ_{wt} - represents the subset of relations Γ_{wt} that links X_w objects with X_t objects, (13) γ_{to} - represents the subset of relations Γ_{to} that links X_t objects with X_o objects, (14) γ_{td} - represents the subset of relations Γ_{td} that links X_t objects with X_o objects. We have numbered these nodes to refer to them in the shape graph of the diagrams.

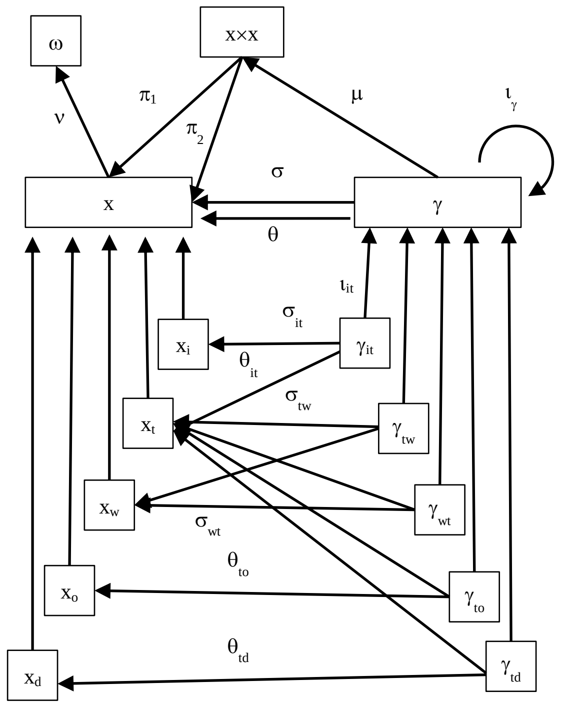

Fig. 9. The Graph of the Sketch.

In the following we will introduce the elements $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$ which impose the constraints specific to our metamodel [7,8] as follows:

1) G is a connected graph. The pushout of σ with θ introduces an equivalence class that defines the set of connected components of the graph [7]. For the graph to be connected we must have only one equivalence class, i.e. the set of equivalence classes is a terminal object in Set.

For this we will introduce into our sketch a cocone K_1. The vertex of this cocone will be ω and the shape graph of this diagram is in Fig. 10. and the functor k_1 corresponding to these diagram is defined as follows: $k_1(9)=\gamma$; $k_1(1)=x$; $k_1(1')=x$; $k_1(\sigma)=\sigma$; $k_1(\theta)=\theta$.

The node denoted with ω in the graph will become the limit of a cone L_1, with an empty base, i.e. a terminal object from Set.

2) There is only one arc between any two nodes. This entails a monomorphism between the set of relations Γ and the set X×X. We will have to define this Cartesian product as the limit of a discrete diagram. We will specify the Cartesian product through the discrete cone L_2.

The graph shape of diagram L_2 is defined by the nodes 1 and 1' and the component functor I_2 is defined as: $I_2(1)=x$; $I_2(1')=x$. The limit of this diagram in Set is the Cartesian product X×X.

The monomorphism $\mu:\Gamma\rightarrow X\times X$ is defined by commutative diagram D_1. Defining the function $\mu:\Gamma\rightarrow X\times X$, can be done by the commutativity of the diagram D_1. The shape graph of this diagram is in Fig. 11. The functor d_1 is defined as follows: $d_1(9)=\gamma$; $d_1(1)=x$; $d_1(7)=x\times x$; $d_1(1')=x$; $d_1(\sigma)=\sigma$;$d_1(\theta)=\theta$; $d_1(\mu)=\mu$; $d_1(\pi_1)=\pi_1$; $d_1(\pi_2)=\pi_2$.

The condition that is required in this commutative diagram to have no more than one arc between any two nodes is that the function μ becomes a monomorphism in Set. But μ is a monomorphism if and only if the pullback of μ with μ exists and is equal to Γ.

$$\begin{tikzcd} 9 \arrow[r, "\sigma"] \arrow[d, "\theta"'] & 1' \\ 1 & \end{tikzcd}$$

Fig. 10. Shape Graph of Pushout Diagram.

$$\begin{tikzcd} & 9 \arrow[dl, "\sigma"'] \arrow[d, "\mu"] \arrow[dr, "\theta"] & \\ 1 & 7 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & 1' \end{tikzcd}$$

Fig. 11. Shape Graph of Commutative Diagram.

$$\begin{tikzcd} & \boxed{9'} \arrow[d, "\mu"] \\ \boxed{9} \arrow[r, "\mu"] & \boxed{7} \end{tikzcd}$$

Fig. 12. Shape Graph of Pullback Diagram.

The pullback of μ with μ is the limit of cone L_3. The shape graph of this diagram is in Fig. 12. and the functor l_3 corresponding to this diagram is defined as: $l_3(9)=\gamma$; $l_3(9')=\gamma$; $l_3(7)=x\times x$; $l_3(\mu)=\mu$. The limit of this diagram in the Set category will have to be Γ.

3) On the set of nodes X we have a partition: $X=X_i \sqcup X_o \sqcup X_d \sqcup X_w \sqcup X_t$. That is, the set of objects X is the disjoint union of five subsets of objects. This means that X is the coproduct of a discrete diagram formed by five nodes and with the vertex X, which in Set will become the colimit of this discrete diagram.

We will specify the partition introducing in the sketch of the model the cocone K_2. The shape graph of this diagram is made up of nodes 3 and 2 and the functor k_2 corresponding to this diagram is defined as: $k_2(2)=x_i$; $k_2(3)=x_o$; $k_2(4)=x_d$; $k_2(5)=x_w$; $k_2(6)=x_t$. The limit of cone K_2 requires that X be the disjoint union of all objects of a model with all adjacent constraints imposed by the other constructs.

4) $\sigma,\theta:\Gamma\rightarrow X$ are functions that associate to an arc, a source and a target. The additional notations σ_{it}, σ_{tw}, σ_{wt}, σ_{to}, σ_{td}, and θ_{it}, θ_{tw}, θ_{wt}, θ_{to}, θ_{td}, will also be reflected in the graph of the sketch because they are operators of the sketch.

Γ is a set of arcs divided into five subsets $\Gamma=\Gamma_{it}\sqcup\Gamma_{tw}\sqcup\Gamma_{wt}\sqcup\Gamma_{to}\cup\Gamma_{td}$. Therefore, the set of arcs Γ is the disjoint union of the five subsets of arcs. This means that Γ is the coproduct of a discrete diagram formed by five nodes.

In the sketch we will specify that X is the colimit of the discrete diagram formed by nodes γ_{it}, γ_{tw}, γ_{wt}, γ_{to}, and γ_{td} through the cocone K_3. The shape graph of this diagram is made up of nodes 10, 11, 12, 13, 14 and the functor k_3 corresponding to this diagram is defined as: $k_3(10)=\gamma_{it}$; $k_3(11)=\gamma_{tw}$; $k_3(12)=\gamma_{wt}$; $k_3(13)=\gamma_{to}$; $k_3(14)=\gamma_{td}$. Therefore, the node denoted with γ in the graph of the sketch will become in the Set category the set Γ which will be the colimit of this discrete diagram.

$|\Gamma x|=1$ for any $x\in X_w$, i.e. the components assembled in a workstation will all go to the same test station. For this we have to make sure that the function $\sigma_{wt}:\Gamma_{wt}\rightarrow X_w$ is a monomorphism, i.e. the pullback of σ_{wt} with σ_{wt} has to be Γ_{wt}. For this we will introduce a cone L_4 in the metamodel sketch. The shape graph of this diagram is in Fig. 13. and the functor l_4 corresponding to this diagram is defined as: $l_4(12)=\gamma_{wt}$; $l_4(12')= \gamma_{wt}$; $l_4(5)= x_w$; $l_4(\sigma_{wt})= \sigma_{wt}$. The limit of this diagram in the Set category will have to be Γ.

So we've got the sketch of a MLMS, we denote it with $L^1(MLMS)=(\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ where: $\mathcal{G}$ is the graph from Fig. 9, $\mathcal{D}=\{D_1\}$, $\mathcal{L}=\{L_1,L_2,L_3,L_4\}$ and $\mathcal{K}=\{K_1,K_2,K_3\}$.

$$\begin{tikzcd} & \boxed{12} \arrow[d, "\sigma"] \\ \boxed{12'} \arrow[r, "\sigma"] & \boxed{5} \end{tikzcd}$$

Fig. 13. Shape Graph of Pullback Diagram.

IV. The Metamodel

A correct model in relation to the sketch $L^1 = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ must be the image of a functor defined on the graph $\mathcal{G}$ in Set which complies with the conditions imposed by the components $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{K}$ of the sketch.

From the way we constructed the L^1 sketch, it follows that this sketch specifies the same mathematical object that is also defined by definition 1. An important advantage of the sketch is that it provides a graphical specification of the metamodel.

We observe two advantages of using the sketch for specifying metamodels, the first is that they are defined in a graphical language and the second is that the constraints imposed by the sketch will be respected by all the models generated on it. The sketches are not designed as notations, but as a mathematical structure incorporating a formal syntax expressed by the semantics of constructions from the category theory.

We note that all the concepts of a model, both the entities involved in the model and the associations between them, are represented by the nodes of the sketch. The arcs of the sketch are not concepts of the model, they are sketch operators and are used to interpret the syntax of the models. These operators will be implemented as algorithms in the metamodel.

The sketch objects that represent the atomic elements of the models will be part of the modeling tool and will then be put on PaletteDrawers to visually serve the models definition procedure.

For this we will define a functor: $\phi:L^1\rightarrow Sets$ that associate to each visual object of the sketch an instance that will be hosted by the modeling tool palette.

Example 2. For example 1, the modeling tool palette will have to host the concepts represented by the nodes of the subsketch from Fig. 14. Only these concepts will serve to visually define the models specified by the sketch L^1.

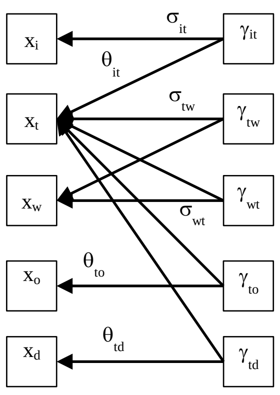

Fig. 14. Subsketch of basic Concepts.

The sketch $L^1 = (\mathcal{G}, \mathcal{D}, \mathcal{L}, \mathcal{K})$ is the MLMS metamodel, that is, the syntax of the modeling language. The image of this sketch through a functor in Set is a model with an imposed syntax. Therefore, specifying a syntactically correct MLSM is equivalent to defining a functor $H^2:L^1 \to Set$ that transforms the sketch nodes into sets of classes that preserve the node type.

The operators of the sketch will be transformed into functions with the same role of operators with the same name and that respect the conditions imposed by the sketch.

Obviously there are models among which we can define a certain similarity of structures in the sense of homomorphism. This similarity is defined by a natural transformation $\tau:H_i^2 \to H_j^2$ which becomes a graph homomorphism between the models generated by the sketch L^1.

If we consider each model H_k^2, $k \geq 0$ generated by the L^1 sketch an object and each natural transformation between two models H_i^2 and H_j^2 as an arrow we will get a category that we call the category of models generated by the sketch L^1 which we denote with Mod(L^1, Set). All models in this category are syntactically correct. The dynamic behavior of a model is given by a sequence of instances associated with each model as we will see in the next section.

V. The Dynamic Behavior of a Process Model

The semantics of a process model represents the dynamic behavior of the modeled real system. This is accomplished by performing procedures according to the interaction rules of the real system components. It is therefore essential that the procedures associated with the model's activities be as faithful as possible to the interaction rules of the components of the real system. The sequence of executed procedures involves the state sequence of the model [12,13,14].

Simulating a process model involves a collection of functions that exploit its knowledge base to enable this information to be treated in such a way as to obtain a similar behavior to that of the simulated system. These functions change the state of the system, i.e. it produces a set of events that in turn determines the execution of other activities [2,3,4].

The basic sketch of a metamodel introduces a series of invariants of a model such as the fact that we cannot have in a model only atomic elements of the types specified by the sketch objects. These invariants make it possible to define the possible states of the model by attribute values at a given time. The transition of the model from one state to another will be done through specific reaction rules called macrotransitions [7,10,11].

A macrotransition that causes the model to pass from the state represented by a vector V^1 to the state represented by a vector V^2 through the reaction rules p can be symbolized by a triplet (V^1, p, V^2).

Since states are often values of the attributes distributed over time, the term event is used instead of the state vector and as a result a macrotransition is symbolized by (E^1, p, E^2) with a meaning similar to the one above.

In the case of our model we will introduce another invariant, namely that any instance of a model will contain only one instance for each object of the model. We can thus define a transition in the form ($\mathfrak{I}^1$, p, $\mathfrak{I}^2$) where $\mathfrak{I}^1$ and $\mathfrak{I}^2$ are instances and p is a natural transformation as we will see below.

In a transition system, if we know the state of the system at one point, we can describe the evolution of the system without the states through which the system passed until it reached its current state [10,12].

We will denote with $L^2 = H^2(L^1)$ a model generated by the L^1 sketch. Each object of the model L^2 is a set of classes that have the corresponding node type of the L^1 sketch. The arcs of the model L^2 represent the model operators.

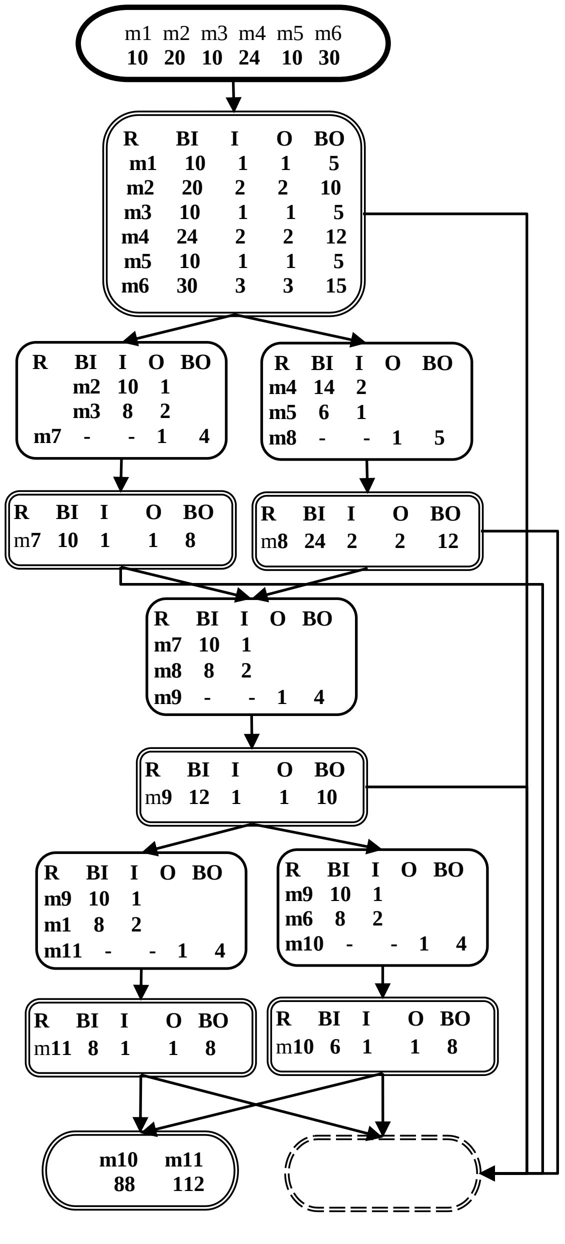

Fig. 15. Example of a MLMS Model.

If we denote with M the graph model corresponding to the model L^2, then $M=J(L^2)$ where J is a functor defined on the set of models generated by the L^1 sketch with values in the graph category $J:Mod(L^1,Set)\rightarrow Graph$. The functor J maps each model to the resulting graph by interpreting sketch operators and natural transformations between models in graph homomorphisms.

Example 3. In example 1, a model L^2 limited to the atomic elements represented in Fig. 14, which are necessary and sufficient for the visual representation of the model, could be that generated by the functor defined as follows:

$H^2(x_i)=X_i=\{I_1\}$ is a set of concepts of type x_i;
$H^2(x_o)=X_o=\{O_1\}$ is a set of concepts of type x_0;
$H^2(x_d)=X_d=\{D_1\}$ is a set of concepts of type x_d;
$H^2(x_w)=X_w=\{W_1, W_2, W_3, W_4, W_5\}$ is a set of concepts of type x_w;
$H^2(x_t)=X_t=\{T_1, T_2, T_3, T_4, T_5, T_6\}$ is a set of concepts of type x_t;
$H^2(\gamma_{it})=\Gamma_{it}=\{\Gamma^1_{it}\}$ represents the subset of relations of type γ_{it};
$H^2(\gamma_{tw})=\Gamma_{tw}=\{\Gamma^1_{tw}, \Gamma^2_{tw}, \Gamma^3_{tw}, \Gamma^4_{tw}, \Gamma^5_{tw}, \Gamma^6_{tw}, \Gamma^7_{tw}, \Gamma^8_{tw}\}$ represents the subset of relations of type γ_{tw};
$H^2(\gamma_{wt})=\Gamma_{wt}=\{\Gamma^1_{wt}, \Gamma^2_{wt}, \Gamma^3_{wt}, \Gamma^4_{wt}, \Gamma^5_{wt}\}$ represents the subset of relations of type γ_{wt};
$H^2(\gamma_{to})=\Gamma_{to}=\{\Gamma^1_{to}, \Gamma^2_{to}\}$ represents the subset of relations of type γ_{to};
$H^2(\gamma_{td})=\Gamma_{td}=\{\Gamma^1_{td}, \Gamma^2_{td}, \Gamma^3_{td}, \Gamma^4_{td}, \Gamma^5_{td}, \Gamma^6_{td}\}$ represents the subset of relations of type γ_{td};
$\sigma_{it}:\Gamma_{it}\rightarrow X_i$ associates to each relation from Γ_{it} the source node from X_i: $\sigma_{it}(\Gamma^1_{it})=I_1$;
$\theta_{it}:\Gamma_{it}\rightarrow X_t$ associates to each relation from Γ_{it} the target node from X_t: $\theta_{it}(\Gamma^1_{it})=T_1$;
$\sigma_{tw}:\Gamma_{tw}\rightarrow X_t$ assigns to each relation from Γ_{tw} the source node in X_t: $\sigma_{tw}(\Gamma^1_{tw})=T_1$; $\sigma_{tw}(\Gamma^2_{tw})=T_1$; $\sigma_{tw}(\Gamma^3_{tw})=T_1$; $\sigma_{tw}(\Gamma^4_{tw})=T_1$; $\sigma_{tw}(\Gamma^5_{tw})=T_2$; $\sigma_{tw}(\Gamma^6_{tw})=T_3$; $\sigma_{tw}(\Gamma^7_{tw})=T_4$; $\sigma_{tw}(\Gamma^8_{tw})=T_4$;
$\theta_{tw}:\Gamma_{tw}\rightarrow X_w$ assigns to each relation from Γ_{tw} the target node in X_w: $\theta_{tw}(\Gamma^1_{tw})=W_1$; $\theta_{tw}(\Gamma^2_{tw})=W_2$; $\theta_{tw}(\Gamma^3_{tw})=W_4$; $\theta_{tw}(\Gamma^4_{tw})=W_5$; $\theta_{tw}(\Gamma^5_{tw})=W_3$; $\theta_{tw}(\Gamma^6_{tw})=W_3$; $\theta_{tw}(\Gamma^7_{tw})=W_4$; $\theta_{tw}(\Gamma^8_{tw})=W_5$;
$\sigma_{wt}:\Gamma_{wt}\rightarrow X_w$ assigns to each relation from Γ_{wt} the source node in X_w: $\sigma_{wt}(\Gamma^1_{wt})=W_1$; $\sigma_{wt}(\Gamma^2_{wt})=W_2$; $\sigma_{wt}(\Gamma^3_{wt})=W_3$; $\sigma_{wt}(\Gamma^4_{wt})=W_4$; $\sigma_{wt}(\Gamma^5_{wt})=W_5$;
$\theta_{wt}:\Gamma_{wt}\rightarrow X_t$ assigns to each relation from Γ_{wt} the target node in X_t: $\theta_{wt}(\Gamma^1_{wt})=T_2$; $\theta_{wt}(\Gamma^2_{wt})=T_3$; $\theta_{wt}(\Gamma^3_{wt})=T_4$; $\theta_{wt}(\Gamma^4_{wt})=T_5$; $\theta_{wt}(\Gamma^5_{wt})=T_6$;
$\sigma_{to}:\Gamma_{to}\rightarrow X_t$ assigns to each relation from Γ_{to} the source node in X_t: $\sigma_{to}(\Gamma^1_{to})=T_5$; $\sigma_{to}(\Gamma^2_{to})=T_6$;
$\theta_{to}:\Gamma_{to}\rightarrow X_o$ assigns to each relation from Γ_{to} the target node in X_o: $\theta_{to}(\Gamma^1_{to})=O_1$; $\theta_{to}(\Gamma^2_{to})=O_1$;
$\sigma_{td}:\Gamma_{td}\rightarrow X_t$ assigns to each relation from Γ_{td} the source node in X_t: $\sigma_{td}(\Gamma^1_{td})=T_1$; $\sigma_{td}(\Gamma^2_{td})=T_2$; $\sigma_{td}(\Gamma^3_{td})=T_3$; $\sigma_{td}(\Gamma^4_{td})=T_4$; $\sigma_{td}(\Gamma^5_{td})=T_5$; $\sigma_{td}(\Gamma^6_{td})=T_6$;
$\theta_{td}:\Gamma_{td}\rightarrow X_d$ assigns to each relation from Γ_{td} the target node in X_d: $\theta_{td}(\Gamma^1_{td})=\theta_{td}(\Gamma^2_{td})=\theta_{td}(\Gamma^3_{td})=\theta_{td}(\Gamma^4_{td})=\theta_{td}(\Gamma^5_{td})=\theta_{td}(\Gamma^6_{td})=D_1$;

Then the MLMS model is like in Fig. 15.

The instantiation of the model L^2 is represented by a functor $\phi:L^2\rightarrow Sets$ that maps each set of classes of the model L^2 to a set of instances of the same type.

Each instance of the model L^2 is a configuration that the process can adopt as a state.

If we now consider every instance of the model L^2 an object and every natural transformation between these instances as an arrow we get a category that we call the CIT category, that is, the category of instances and natural transformations of the model L^2.

The CIT category offers the contextual routes of evolution of a process model and represents the possible interactions between the process and its environment. It is often possible to analyze the dynamics of a process only through the transitions offered by the CIT category. But the execution of the model will be based on the reaction rules specific to each process in the context of admissible routes from the CIT category. The set of reaction rules determines a reaction relation between the admissible states that are represented by the instances of the model [11].

A process configuration is a state of the process and is characterized by the values of the attributes associated to each atomic element, as well as by its structure within the boundaries offered by the natural transformations that coincide with homomorphisms between the corresponding graphical models. Thus, the macrotransitions resulting from this combination represent the actual behavior of the system modeled in interaction with a given context [10,11,14].

If the configuration $\mathfrak{I}'$ is obtained from the configuration $\mathfrak{I}$ by applying the reaction rules of the model, then we say that between $\mathfrak{I}$ and $\mathfrak{I}'$ there is a reaction relation from $\mathfrak{I}$ to $\mathfrak{I}'$ which we denote by $\mathfrak{I}\Rightarrow\mathfrak{I}'$. If the application of the reaction rules is done in the context of a natural transformation τ then we will denote this with $\mathfrak{I}\xrightarrow{\tau}\mathfrak{I}'$.

In this context, the execution of a process becomes a sequence of reaction rules in the context of natural transformations, a path in the CIT category.

$$\mathfrak{I}_0\xrightarrow{\tau^0}\mathfrak{I}_1\xrightarrow{\tau^1}\ldots\mathfrak{I}_n\xrightarrow{\tau^n}\ldots$$

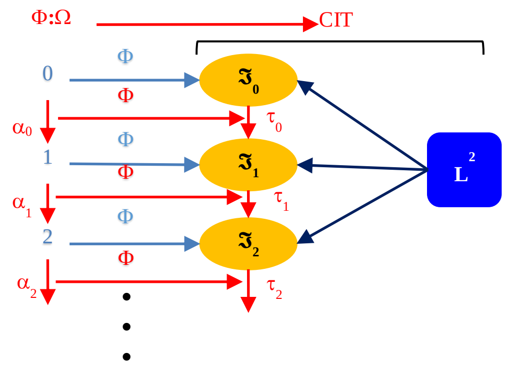

Fig. 16. Execution of a Model.

Of course that each model can support a set of different executions in different contexts. All these executions can be specified by functors that map the category Ω defined as follows:

$\Omega: 0 \overset{\alpha_0}{\to} 1 \overset{\alpha_1}{\to} \ldots k \overset{\alpha_k}{\to} \ldots$

In the CIT category. Each corresponding functor Φ:Ω→CIT orders a set of model configurations over time and is defined as follows (Fig. 16)

$\Phi(i) = \mathfrak{I}_i$ for all $i \geq 0$; $\Phi(\alpha_i) = \tau^i$ for all $i \geq 0$

Example 4. In example 1 the state of an instance is given by the values of the attributes representing the current quantities in input buffers X_i (entry stations), output buffers X_O (exit station), faulty components buffer X_d, and respectively in the input and output buffers of every assembling and testing stations X_w and X_t. Natural transformations between model instances are represented by macro-transitions triggered by the state of each instance. The trigger rules are specific to each concept. A transport action, represented by the arcs of the model, is performed if at the source buffer there are at least as many parts as the transport capacity and in the exit buffer there is enough storage space. An assembly action triggers if all the required parts are in the input buffer and there is storage capacity in the output buffer. A test action is triggered if there are necessary parts in the entry buffer, and there is storage capacity in the output buffer.

In order to optimize the workflow, execution time proportional to real time was included for each action.

The MLMS metamodel was implemented in MM-DSL then translated and executed in ADOxx. Also, to demonstrate the concept, we also implemented the PN grammar.

VI. Conclusion

In this paper we presented a new approach based on the category theory in specifying metamodels. The visual specification of the model facilitates the involvement of specific field experts in the metamodel specification and validation process.

There is a natural link between process models and category theory. The category theory provides a representation of a metamodel as a mathematical object, a sketch, and a model as a functor that is also a mathematical object, thus simplifying the way to think of a process model. In this way, all theoretical results in the category theory can help solve some classic problems in modeling processes.

A modeling tool is more than a programming language, it can be understood as a modeling method [1]. A modeling tool usually contains a visual language and a set of mechanisms and algorithms.

In the categorical model from this paper the grammar of the language is specified by a categorical sketch. Mechanisms and algorithms represented by natural transformations. Universal constructions in the category theory allow for the implementation of mechanisms and algorithms with a high degree of generality at the level of the metamodels.

It is not too hard to see that defining models as functors creates a framework for addressing model migration issues and multi-paradigm modeling. We will address these issues in future papers.

The CIT category represents the admissible routes of any process at the metamodel level. Of course these routes are then conditioned and thus validated or invalidated by the execution rules of each model.

The natural transformations from the CIT category are objects that also have a certain state and can therefore aggregate information about process progress over time, execution frequency of each activity, execution times, costs, etc. Also, these objects can be endowed with artificial intelligence to make decisions and learn the best performing routes in relation to various criteria. All these features can be implemented at the metamodel level.

References

[1] 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.

[2] Dimitris Karagiannis, H. Kühn, 2002. *Metamodelling Platforms.* Invited paper in: Bauknecht, K.; Tjoa, A Min.; Quirchmayer, G. (eds.): Proceedings of the Third International Conference EC- Web 2002 - Dexa 2002, Aix-en-Provence, France, September 2-6, 2002, LNCS 2455, Springer-Verlag, Berlin, Heidelberg.

[3] 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.

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

[5] Dimitris Karagiannis, Junginger S., Strobl R., 1996. *Introduction to Business Process Management Systems Concepts.* In: Scholz-Reiter B., Stickel E. (eds) Business Process Modelling. Springer, Berlin, Heidelberg, 1996

[6] Ernest G. Manes, Michael A. Arbib, 1986. *Algebraic Approaches to program semantics*, Springer Verlag New York Berlin Heidelberg London Paris Tokyo – 1986.

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

[8] Michael Barr, Charles Wells, 2002. *Toposes, Triples and Theories*, November 2002.

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

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

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

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

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

[14] W.M.P. van der Aalst and K.M. van Hee, 2004. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2004.