DOI: 10.2478/ijasitels-2025-0011

Integrating Multi-View features into diagrammatic languages

Daniel C. Crăciunean[1]

[1]*Computer Science and Electrical and Electronics Engineering Department, Faculty of Engineering, "Lucian Blaga" University of Sibiu, Romania*
daniel.craciunean@ulbsibiu.ro

Abstract

In this paper, we define diagrammatic languages by metamodels represented by categorical sketches. Then, a model, corresponding to a categorical sketch, is a Set-value functor M:$\mathcal{G}$→Set, where $\mathcal{G}$ is the sketch graph, and which respects the constraints imposed by the categorical sketch. A view of a model M, is another model $\mathcal{V}$ in the diagrammatic language that maps to M by a natural injective transformation. This is a theoretically satisfactory definition but does not provide us with solutions to actually find these views. In this paper, we will present a practical way to construct a satisfactory set of views for a given model. Moreover, this mechanism can be implemented generically, at the modeling language level, and then used for each model separately. The categorical mechanisms that we use to solve this problem are representable functors and the Yoneda lemma. This is also the main contribution of this paper to solving problems related to multi-view modeling.

Keywords: category, functor, model, categorical sketch, representable functors, multi-view, meta-model

1 Introduction

The high complexity of the systems that modeling activity currently faces requires the decomposition of models according to various complementary points of view. A conceptual and methodological approach [9], which allows modeling a complex system by decomposing it into complementary sub-models, is multi-view modeling [10]. In this approach, each view captures, partially or totally, a certain structural, behavioral or functional aspect of the system. These sub-models can be assembled into a whole that provides a complete and coherent picture of the modeled system.

Most modern complex systems such as cyber-physical or socio-technical systems involve multiple dimensions and actors with different concerns, objectives and perspectives. Under these conditions, a single global model rarely manages to capture all the necessary requirements. Multi-view modeling can respond to these challenges by conceptually decomposing the system into several sub-models, depending on these requirements.

In the Model-Driven Engineering (MDE) paradigm, views are defined as independent models, which are connected through mappings, transformations or synchronization rules that make up the concept of Model Integration. In the context of multi-level

modeling, hierarchical levels of abstraction can be detailed through views, thus maintaining coherence between levels and views.

Significant conceptual flexibility offered by multi-view modeling, however, also comes with a series of challenges related, first of all, to maintaining inter-model consistency, especially in large, dynamic systems and to automatically synchronizing changes between views. In the case of linguistically heterogeneous models, semantic compatibility issues also arise. In general, the solutions proposed by several recent researches to solve these problems are based on model transformation [4], ontology-based integration and co-simulation [15][8].

In the MDE paradigm, the development of domain-specific diagrammatic modeling languages [14] is an integral part of the modeling process [11]. Obviously, in the context of multi-view modeling, these languages must be endowed with special facilities that simplify the multi-view modeling process [10].

In this paper, we define diagrammatic languages by metamodels represented by categorical sketches. This approach allows the use of categorical concepts such as representable functors and Yoneda's lemma to endow diagrammatic languages with important facilities for specifying the views of a model. Thus, the endowment of diagrammatic modeling languages with multi-view modeling facilities based on representable functors and Yoneda's lemma represents the main contribution of this paper to the field of multi-view modeling.

Section 2 of the paper contains background notions, section 3 defines diagrammatic languages based on a meta-model represented by a categorical sketch, section 4 presents the use of representable functors in integrating multi-view modeling facilities into diagrammatic languages. In the last section, some conclusions and observations are presented.

2 Background Notions

Graph theory and category theory are the fundamental theories that support modeling in general and multi-view modeling in particular. In this section, we will briefly present some notions in this field and some notations used in this paper.

A graph is a tuple $\boldsymbol{G}=(X, \Gamma, \sigma, \theta)$ composed of the set of nodes X, the set of relations Γ and two functions $\sigma,\theta:\Gamma\rightarrow X$ indicating the source and, respectively, the target of each arc of the graph. Intuitively, a graph is a collection of nodes connected by directed arcs, but without composition rules. In its most intuitive form, a category is a graph with rules for the composition of arrows and with identity arrows for each node.

Thus, a category $\boldsymbol{C}$ consists of a set of objects $ob(\boldsymbol{C})$ and for each pair of objects A,B, a set of morphisms (arrows) Hom(A,B). The set of morphisms is endowed with a partial composition operation: $o:Hom(B,C)\times Hom(A,B)\rightarrow Hom(A,C)$ which is associative and for each object A, there is an identity morphism $id_A\in Hom(A,A)$ [1][5].

As we see, a category is endowed with a structure, given by the composition operation, unlike graphs. This structure also leads to the difference between a graph homomorphism and a functor. Thus, a functor is an application between categories. Which takes objects into objects and arrows into arrows and which preserves the categorical structure.

Among all functors, Set-value functors are distinguished, which are useful in modeling. A Set-value functor is a functor that maps an arbitrary category $\boldsymbol{\mathcal{C}}$ into the category Set [1] [5].

Between two functors ϕ and ψ, from the category $\boldsymbol{\mathcal{C}}$ to the category $\boldsymbol{\mathcal{D}}$, we can define a map τ, which maps the image of the functor ϕ onto the image of the functor ψ, in the category $\boldsymbol{\mathcal{D}}$, and which if it satisfies a condition, called the naturalness condition, is called a natural transformation. Natural transformations are defined at the level of objects of the category $\boldsymbol{\mathcal{C}}$. The naturalness condition requires that for any arrow f:A→B in $\boldsymbol{\mathcal{C}}$, we have the equality $(\psi f) \circ \tau_A = \tau_B \circ (\phi f)$, where τ_A and τ_B are the components of τ for objects A and B [1] [2].

Natural transformations are maps that express a natural way of mapping one functor to another. Obviously, natural transformations can be composed by the classical operation of composition of functions. Therefore, if we have two categories $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$, we can form a category of functors that has as objects: functors defined on $\boldsymbol{\mathcal{C}}$ with values in $\boldsymbol{\mathcal{D}}$, as arrows the natural transformations between functors and as composition operation, the composition of natural transformations on components.

In this paper we specify diagrammatic languages by metamodels represented by categorical sketches. A categorical sketch [13], is a tuple formed by a graph $\boldsymbol{\mathcal{G}}$, and a set of constraints $C(\boldsymbol{\mathcal{G}})$, on models expressed by Set-value functors.

Constraints can be imposed by commutative diagrams, categorical limits and colimits or by predicates expressed on the components of some diagrams, called graph forms [7] [12] [13]. Graph forms are used as diagrams on which conditions are imposed by commutativity, limits or colimits, or as shape graph arity for predicates that specify constraints.

3 Diagrammatic languages

We define diagrammatic languages by meta-models represented by categorical sketches. A categorical sketch is composed of a typed graph $\boldsymbol{\mathcal{G}}$ and a set of constraints $C(\boldsymbol{\mathcal{G}})$ imposed on the image of the graph $\boldsymbol{\mathcal{G}}$, by a functor M in the category Set, that is, on a model represented by the sketch $\boldsymbol{\mathcal{S}}=(\boldsymbol{\mathcal{G}}, \boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{G}}))$ [6] [7].

The typed graph $\boldsymbol{\mathcal{G}}=(T, \Gamma, \sigma, \theta)$ is composed of the set of node types T, the set of relationship types Γ and two functions $\sigma,\theta:\Gamma\rightarrow T$ indicating the source and, respectively, the target of each arc of the graph. Obviously, the sets T and Γ can represent both concept types from the modeling domain and attribute types attached to the concepts.

A model $M:\boldsymbol{\mathcal{S}}\rightarrow S$, is a functor $M:\boldsymbol{\mathcal{G}}\rightarrow Set$, which maps each type of object $t\in T$ into a set of objects of type t and each type of arc $\gamma\in\Gamma$ into a function of the same type as the corresponding arc [6] [7].

Example 1. We will consider a very simple example of a categorical sketch to illustrate our approach to the problems treated in this paper. First, we define the types of concepts in the modeling domain and the graphical atoms of the modeling language that we want to specify. We therefore have the following concepts:

A type T_1 that represents buffers with limited capacity, which can store a single type of elements, and for which we use as graphical notation a red circle.

The type T_1 will also have two attributes, namely: stock and capacity.

A type T_2 that represents buffers with limited capacity, which can store two types of elements, and for which we use as graphical notation a blue rectangle.

The type T_2 will also have four attributes, namely: stock1, stock2, capacity1 and capacity2.

A type T_3 that contains an integer from a range of integers, representing values for the attributes of types T_1 and T_2, and for which we use a white circle as a graphic notation.

The type T_3 will have an attribute, namely: value.

We will also use oriented arrows to represent the relationships and functions involved in the diagrams. For our example, we will consider that the graph $\boldsymbol{\mathcal{G}}$, of the sketch $\boldsymbol{\mathcal{S}}$ is the one in Figure 1. To simplify the approach, we will consider that the sketch $\boldsymbol{\mathcal{S}}$, which is our metamodel, does not contain constraints. We also mention that the sketch graph could represent the attributes and even the algebra generated by the attributes [3] [4].

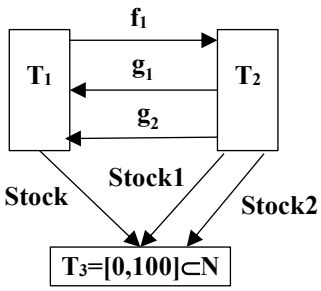

Figure 1. The graph of the sketch

A model specified by the sketch $\boldsymbol{\mathcal{S}}$ is the following: $M(T_1)=\{D_{11},D_{12},D_{13},D_{14}\}$; $M(T_2)=\{D_{21},D_{22}\}$; $M(T_3)=\{10,20,30\}$;

Although the arcs of the graph become functions, we will write them with the same symbols, that is, we consider that: $M(f_1)=f_1$; $M(g_1)=g_1$; M(Stock)=Stock; M(Stock1)=Stock1; M(Stock2)=Stock2.

These are defined as follows:

f_1: $M(T_1)\rightarrow M(T_2)$; $f_1(D_{11})=D_{22}$, $f_1(D_{12})=D_{22}$, $f_1(D_{13})=$Null, $f_1(D_{14})=$Null,

Stock: $M(T_1)\rightarrow M(T_3)$; $Stock(D_{11})=20$, $Stock(D_{12})=10$, $Stock(D_{13})=30$, $Stock(D_{14})=30$,

g_1: $M(T_2)\rightarrow M(T_1)$; $g_1(D_{21})=D_{11}$, $g_1(D_{22})=D_{13}$,

g_2: $M(T_2)\rightarrow M(T_1)$; $g_2(D_{21})=D_{12}$, $g_2(D_{22})=D_{14}$,

Stock1: $M(T_2)\rightarrow M(T_3)$; $Stock1(D_{21})=20$, $Stock1(D_{22})=10$,

Stock2: $M(T_2)\rightarrow M(T_3)$; $Stock2(D_{21})=10$, $Stock2(D_{22})=20$,

This model can be seen in diagrammatic form in Figure 2.

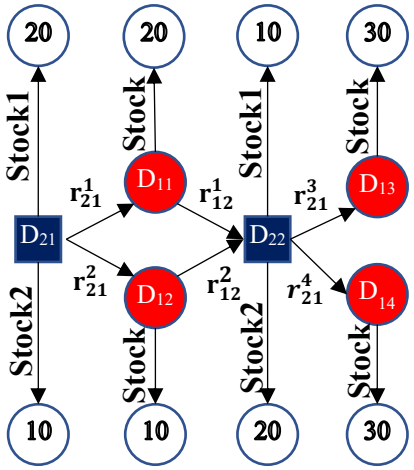

Figure 2. An instance of the sketch in Figure 1.

The diagrammatic language DL, represented by the sketch $\boldsymbol{\mathcal{S}}$, consists of all functors of the form M:$\boldsymbol{\mathcal{S}}$→Set and we denote it by DL($\boldsymbol{\mathcal{S}}$).

If $\boldsymbol{\mathcal{S}}$ is a categorical sketch and M_1, M_2 are two models, that is, two functors M_1,M_2:$\boldsymbol{\mathcal{S}}$→Set, then we can define a natural transformation τ:M_1,→M_2. The set of all functors M:$\boldsymbol{\mathcal{S}}$→Set together with the natural transformations between them form a category that we denote by $\boldsymbol{\mathcal{M}}(\boldsymbol{\mathcal{S}})$ and we call it the category of models of the language DL($\boldsymbol{\mathcal{S}}$).

As we mentioned, in a diagrammatic language it is important to have mechanisms to specify a variety of views. Obviously, these views are components of the language, that is, they are a specific part of a diagrammatic language.

In our approach, a view of the diagrammatic language DL($\boldsymbol{\mathcal{S}}$) over a model M∈DL($\boldsymbol{\mathcal{S}}$) is a model $\boldsymbol{\mathcal{V}}$∈DL($\boldsymbol{\mathcal{S}}$) with the property that there is a natural injective transformation τ:$\boldsymbol{\mathcal{V}}$→M. It is obvious that the model $\boldsymbol{\mathcal{V}}$ represents a coherent part of the model M.

To manage and manipulate views, we need two very important operations with views, namely, the intersection of two views that allows the visualization of their common part and the union of two views that allows the visualization of the content that cumulates the two views.

Example 2. In Figure 3 and Figure 4 we have two different views of the model in Figure 2.

In our approach we will use pullback for the intersection operation and pushout for the union operation. Pullback is the limit of a co-span in category theory and represents the generalized intersection of two objects over a common object [1]. Pushout is the co-limit of a span in category theory and represents the generalized joining of two objects over a common sub-object [1].

Let M∈DL($\boldsymbol{\mathcal{S}}$) and $\boldsymbol{\mathcal{V}}_1$, $\boldsymbol{\mathcal{V}}_2$ ∈DL($\boldsymbol{\mathcal{S}}$) be two views and τ_1,τ_2 be the two natural injective transformations τ_1: $\boldsymbol{\mathcal{V}}_1$ →M and τ_2: $\boldsymbol{\mathcal{V}}_2$→M. The pullback of $\boldsymbol{\mathcal{V}}_1$ with $\boldsymbol{\mathcal{V}}_2$ is the categorical limit of the co-span, $\boldsymbol{\mathcal{V}}_1 \xrightarrow{\tau_1}M\xleftarrow{\tau_2}$ $\boldsymbol{\mathcal{V}}_2$, which is the view of $\boldsymbol{\mathcal{V}}$, which makes the diagram in Figure 5 commutative.

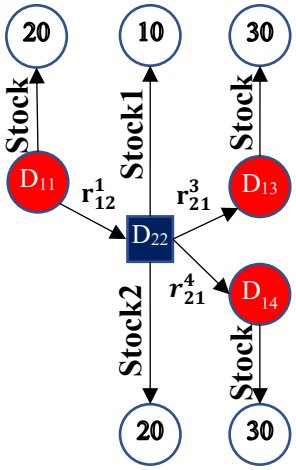

Figure 3. A view instance of the model in Figure 2.

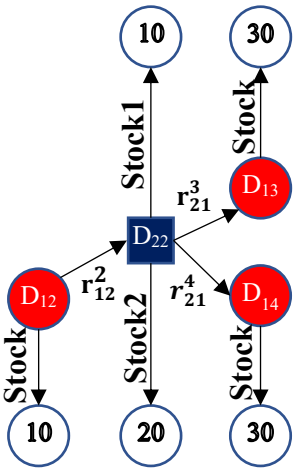

Figure 4. A view instance of the model in Figure 2.

It is worth noting that the pullback preserves injective homomorphisms [1], that is, in our case, it transmits the injectivity property to the resulting homomorphism $\tau:\boldsymbol{\mathcal{V}}\rightarrow M$, where $\tau = \tau_1 o \iota_1 = \tau_2 o \iota_2$.

Let $M \in DL((\boldsymbol{\mathcal{S}})$ and $\boldsymbol{\mathcal{V}}_0, \boldsymbol{\mathcal{V}}_1, \boldsymbol{\mathcal{V}}_2 \in DL(\boldsymbol{\mathcal{S}})$ be three views and τ_0, τ_1, τ_2 be the three natural injective transformations τ_0: $\boldsymbol{\mathcal{V}}_0 \rightarrow M$, τ_1: $\boldsymbol{\mathcal{V}}_1 \rightarrow M$ and τ_2: $\boldsymbol{\mathcal{V}}_2 \rightarrow M$. If the view $\boldsymbol{\mathcal{V}}_0$ is a common subobject of the views $\boldsymbol{\mathcal{V}}_1$ and $\boldsymbol{\mathcal{V}}_2$, then the pushout of $\boldsymbol{\mathcal{V}}_1$ with $\boldsymbol{\mathcal{V}}_2$ is the categorical co-limit of the span, $\boldsymbol{\mathcal{V}}_1 \xleftarrow{\iota_1} \boldsymbol{\mathcal{V}}_0 \xrightarrow{\iota_2} \boldsymbol{\mathcal{V}}_2$, which is the view $\boldsymbol{\mathcal{V}}$, which makes the diagram in Figure 6 commutative. In this span ι_1 and ι_2 are the inclusion applications.

It is worth noting that pushout preserves injective homomorphisms [1], that is, in our case, it transmits the injectivity property to the resulting homomorphism $\mu:\boldsymbol{\mathcal{V}}_0 \rightarrow \boldsymbol{\mathcal{V}}$, where $\mu = \mu_1 o \iota_1 = \mu_2 o \iota_2$. Therefore, the resulting model $\boldsymbol{\mathcal{V}}$ is also a view of M.

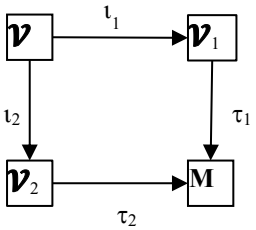

Figure 5. Pullback diagram

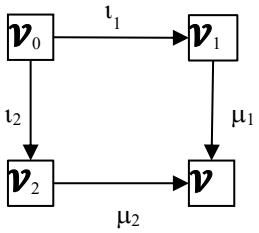

Figure 6. Pushout diagram

Example 3. In Figure 7 we have the pullback of the views in Figures 3 and 4, and in Figure 8 we have the pushout of these models.

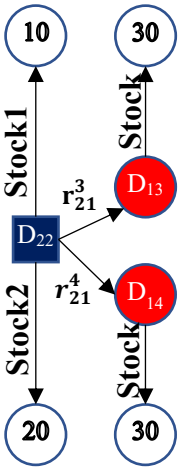

Figure 7. Pullback of the views of Figure 3 and 4

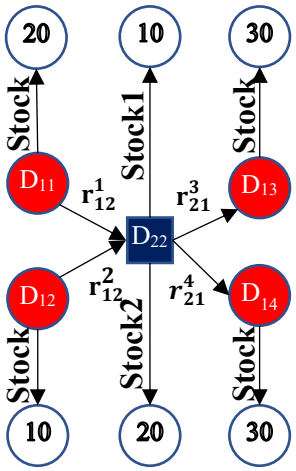

Figure 8. Pushout of the views of Figure 3 and 4

4 Integrating multi-view modeling facilities into diagrammatic languages

In the previous section we saw that a view of a model M is another model $\boldsymbol{\mathcal{V}}$ in the diagrammatic language that maps via an injective homomorphism to M. This is a theoretically satisfactory definition but does not provide us with any solution to actually find these views. In the remainder of this section, we will present a practical way to construct a satisfactory set of views for a given model. Moreover, this mechanism can be implemented generically, at the level of a modeling language, and then used for each model separately. The categorical mechanisms that underlie our approach are representable functors and the Yoneda lemma.

If $\boldsymbol{\mathcal{C}}$ is a category then there is a set of special functors defined on $\boldsymbol{\mathcal{C}}$ with values in Set that are completely determined by the objects in $\boldsymbol{\mathcal{C}}$. More precisely, for every object C∈ob($\boldsymbol{\mathcal{C}}$) there is a set-valued functor represented by C. This functor is Hom$_{\boldsymbol{\mathcal{C}}}$(C,-): $\boldsymbol{\mathcal{C}}$→Set, and is defined as follows: For every object A∈$\boldsymbol{\mathcal{C}}$, we have Hom$_{\boldsymbol{\mathcal{C}}}$(C,-)(A)= Hom$_{\boldsymbol{\mathcal{C}}}$(C,A) and for every arrow f:A→B, we have Hom$_{\boldsymbol{\mathcal{C}}}$(C,-)(f)=Hom$_{\boldsymbol{\mathcal{C}}}$(C,f) where Hom$_{\boldsymbol{\mathcal{C}}}$(C,f)(g)=fog.

We will denote the functor corresponding to the object C∈$\boldsymbol{\mathcal{C}}$ by Y_C. Therefore, we have Y_C(A)=Hom$_{\boldsymbol{\mathcal{C}}}$(C,A) for all objects A∈ob($\boldsymbol{\mathcal{C}}$) and Y_C(f)= Hom$_{\boldsymbol{\mathcal{C}}}$(C,f), for all arrows f in $\boldsymbol{\mathcal{C}}$. It is obvious that if Hom$_{\boldsymbol{\mathcal{C}}}$(C,A)= Y_C(A), is an object in Set and g∈Hom$_{\boldsymbol{\mathcal{C}}}$(C,A) then Y_C(f)(g) = fog∈Hom$_{\boldsymbol{\mathcal{C}}}$(C,B)= Y_C(A) which is an object in Set.

If C is an object from category $\boldsymbol{\mathcal{C}}$, then the functor Y_C is called the functor represented by C and any functor isomorphic to Y_C is called a representable functor.

In order to specify the representable functors, in the case of models, for the sketch $\boldsymbol{\mathcal{S}}$, we will consider the free category generated by the graph $\boldsymbol{\mathcal{G}}$ of the sketch. It is known that a graph can generate a category called free category, which has as objects the nodes of the graph and as arrows the arcs of the graph. The identities are given by the identity arcs attached to each node and the composition of the arrows is given by the concatenation of the paths in the graph.

Example 4. We consider the free category generated by the sketch graph $\boldsymbol{\mathcal{S}}$ from Figure 1. For a more concise presentation we will denote it by cu f^n=f o f o…o f, i.e. f^0=id and f^n=f^{n-1} o f. With this notation the functor represented by T_1, is Y_{T_1}=Hom(T_1,-) defined on components.

1) Each object in $\boldsymbol{\mathcal{G}}$ will be mapped to a set of arcs in the Set category:

$Y_{T_1}(T_1)=\{id_{T_1}\}\cup\{(g_1 \text{ o } f_1)^n, \mid n\geq 0\}\cup\{(g_2 \text{ o } f_1)^n \mid n\geq 0\}$,

$Y_{T_1}(T_2)=\{f_1, \text{Stock}\}\cup\{(f_1 \text{ o } g_1)^n \mid n\geq 0\}\cup\{(f_1 \text{ o } g_2)^n \mid n\geq 0\}$,

$Y_{T_1}(T_3)=\{\text{Stock}, \text{Stock1 o } f_1, \text{Stock2 o } f_1\}$.

2) Each arc in $\boldsymbol{\mathcal{G}}$ will be mapped to a function that maps functions to functions in the Set category as follows:

The function $Y_{T_1}(f_1)$=Hom(T_1,f_1): $Y_{T_1}(T_1)\rightarrow Y_{T_1}(T_2)$ maps each function g∈$Y_{T_1}(T_1)$ to the function $Y_{T_1}(f_1)(g)=f_1$ o g.

The function $Y_{T_1}(g_1)$=Hom(T_1,g_1): $Y_{T_1}(T_2)\rightarrow Y_{T_1}(T_1)$ maps each function g∈$Y_{T_1}(T_2)$ to the function $Y_{T_1}(g_1)(g)=g_1$ o g.

The function $Y_{T_1}(g_2)=\text{Hom}(T_1,g_2)$: $Y_{T_1}(T_2)\rightarrow Y_{T_1}(T_1)$ maps each function $g\in Y_{T_1}(T_2)$ to function $Y_{T_1}(g_2)(g)=g_2 \circ g$.

The function $Y_{T_1}(\text{Stock})=\text{Hom}(T_1,\text{Stock})$: $Y_{T_1}(T_1)\rightarrow Y_{T_1}(T_3)$ maps each function $g\in Y_{T_1}(T_1)$ to function $Y_{T_1}(\text{Stock})(g)=\text{Stock} \circ g$.

The function $Y_{T_1}(\text{Stock1})=\text{Hom}(T_1,\text{Stock1})$: $Y_{T_1}(T_2)\rightarrow Y_{T_1}(T_3)$ maps each function $g\in Y_{T_1}(T_2)$ to function $Y_{T_1}(\text{Stock1})(g)=\text{Stock1} \circ g$.

The function $Y_{T_1}(\text{Stock2})=\text{Hom}(T_1,\text{Stock2})$: $Y_{T_1}(T_2)\rightarrow Y_{T_1}(T_3)$ maps each function $g\in Y_{T_1}(T_2)$ into the function $Y_{T_1}(\text{Stock2})(g)=\text{Stock2} \circ g$.

We note that in the case of a diagrammatic language a functor represented by a certain type represents an abstract model that generically specifies the attributes of the model. We will show in this paper that these functors become views for the concrete models specified by the sketch $\boldsymbol{\mathcal{S}}$. For this we will take advantage of one of the most important results in category theory, namely Yoneda's lemma.

Intuitively, Yoneda's lemma says that any object in a category is completely determined, up to an isomorphism, by its interaction through arrows with the other objects of the category, that is, by the structure of the category.

Specifically, Yoneda's lemma states that if $\boldsymbol{\mathcal{C}}$ is a category, C an object of $\boldsymbol{\mathcal{C}}$, and Ψ:C→Set a set-valued functor, then there is a bijection between the set of homomorphisms from Y_C to Ψ and the set $\Psi(C)$, namely $\text{Hom}(Y_C,\Psi)\xrightarrow{\cong}\Psi(C)$ [5] [1].

We can see that the lemma refers to natural transformations and says that defining a natural transformation from Y_C to Ψ, is equivalent to choosing a concrete element from the set $\Psi(C)$. In the case of models, if $M:\boldsymbol{\mathcal{G}}\rightarrow$Set, and C an object of the free category generated by $\boldsymbol{\mathcal{G}}$, then for each element $c\in M(C)$ there is a unique injective natural transformation $h_c:Y_C\rightarrow M$. Therefore, Y_C is a generic view for the model M.

Example 5. The generic models specified by the sketch $\boldsymbol{\mathcal{S}}$ and the functor Y_{T_1} have the diagrammatic form in Figure 9. We now consider again the concrete model in Example 1, where: $M(T_1)=\{D_{11},D_{12},D_{13},D_{14}\}$; $M(T_2)=\{D_{21},D_{22}\}$; $M(T_3)=\{10,20,30\}$; $M(f_1)=f_1$; $M(g_1)=g_1$; M(Stock)= Stock; M(Stock1)= Stock1; M(Stock2)= Stock2.

We observe that $M(T_1)$ has four elements and therefore it results, according to Yoneda's lemma, that we will have four views of type Y_{T_1}, one for each value given to $x\in M(T_1)$, in the generic model in Figure 9. Thus, for example, for $x=D_{11}$, we have the view in Figure 3, for $x=D_{12}$, we have the view in Figure 4.

Multi-view modeling does not only involve decomposing a model into several sub-models, but also a rigorous mechanism for managing the relationships between these sub-models so that the integrated model remains consistent. As we can see in our approach, the fact that each view is in turn a model of the categorical sketch guarantees the consistency of the integrated model and ensures, at the same time, great flexibility for the sub-models.

The example we used to illustrate our demersal is a very simple one. Obviously, in relation to multi-view modeling, the separation of concerns is also pursued and therefore views can represent not only structural sub-models but also behavioral or functional sub-models. Of course, the approach in these cases does not differ much in the context in which all these aspects will be defined by a categorical sketch and Set-value functors.

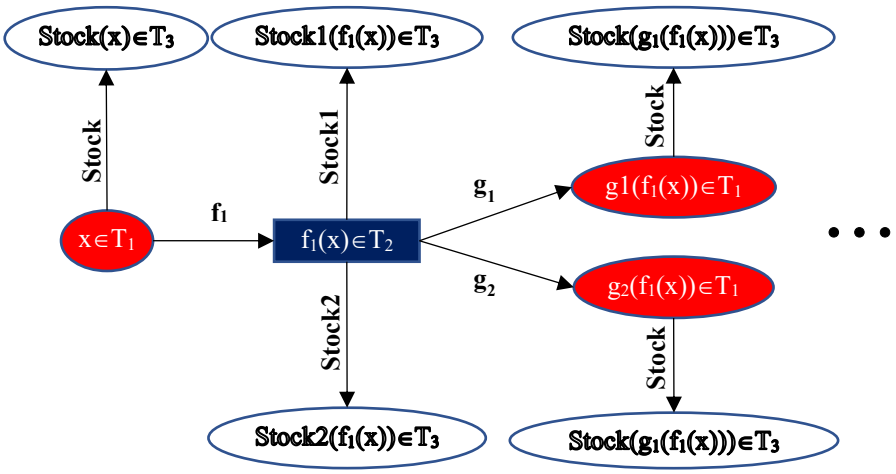

Figure 9. A Y_{T_1} instance of the sketch in Figure 1.

5 Observations and conclusions

Multi-view modeling provides a robust conceptual framework for modeling, specifying, and analyzing complex systems by separating concerns and integrating multiple perspectives coherently. Multi-view modeling is an essential pillar in issues related to the decomposition of systems into subsystems, subsystem interoperability, consistency between sub-models, and integration of sub-models of a complex system.

Modern systems modeling, characterized by structural complexity and heterogeneity, requires the use of powerful formal mechanisms that provide advanced modeling and analysis facilities. In this context, integrated MDE methodologies require more powerful mathematical tools than graph theory. Category theory can provide the context and mathematical mechanisms necessary to solve many modeling problems.

Along with multi-level modeling and co-simulation, multi-view modeling emerges as a complementary pillar of an integrated MDE methodology. Therefore, the use of categorical mechanisms to facilitate multi-view modeling is welcome.

This paper demonstrates that defining diagrammatic languages through metamodels represented by categorical sketches allows the use of important categorical concepts such as representable functors and Yoneda's lemma to endow the diagrammatic language with special facilities for specifying the views of a model.

The use of categorical mechanisms for defining diagrammatic languages through metamodels allows the use of all important theoretical results such as universal constructions, representable functors, Yoneda's lemma and many others obtained in decades of research. The synergy between these mechanisms generates a unified modeling and analysis framework, capable of managing the complexity of cyber-physical systems, flexible manufacturing processes and other domains where interactions between components are critical.

References

[1] Michael Barr, Charles Wells, “Category Theory for Computing Science”, Reprints in Theory and Applications of Categories, No. 22, (2012).
[2] Walters R. F. C., “Categories and Computer Science”, Cambridge Texts in Computer Science, Edited by D. J. Cooke, Loughborough University, (2006).
[3] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel and Ulrike Prange, “Consistent integration of models based on views of meta models in Formal Aspects of Computing”, 22, 327–344, Springer, (2010).
[4] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, Frank Hermann, “Graph and Model Transformation General Framework and Applications”, Springer-Verlag Berlin Heidelberg, (2015).
[5] David I. Spivak, “Category Theory for the Sciences”, The MIT Press Cambridge, Massachusetts London, England, Massachusetts Institute of Technology, (2014).
[6] Daniel-Cristian Craciunean, Daniel Volovici, “Conceptualization of Modeling Method in the Context of Categorical Mechanism”, in Dimitris Karagiannis and others, Domain Specific Conceptual Modeling, Springer Nature Switzerland AG (2022).
[7] Daniel-Cristian Craciunean, “Categorical Mechanisms in Multi-level Modeling Methods”, Publishing House of the "Lucian Blaga" University of Sibiu, (2023).
[8] Daniel-Cristian Crăciunean, D. Karagiannis, “A categorical model of process cosimulation”, Journal of Advanced Computer Science and Applications (IJACSA), 10, (2019).
[9] D. Karagiannis, H.C. Mayr, J. Mylopoulos, “Domain-Specific Conceptual Modeling Concepts, Methods and Tools”, Springer International Publishing Switzerland, (2016).
[10] Kühn, E., & Karagiannis, D., “A multi-view modeling approach for complex systems”, Complex Systems Informatics and Modeling Quarterly, (24), 39–60, (2020).
[11] Dominik Bork , Dimitris Karagiannis, Benedikt Pittl, “A survey of modeling language specification techniques”, Information Systems 87 101425, journal homepage: www.elsevier.com/locate/is. , (2020).
[12] Zinovy Diskin, Uwe Wolter, “A Diagrammatic Logic for Object-Oriented Visual Modeling”, Electronic Notes in Theoretical Computer Science, Volume 203, Issue 6, 21 November (2008).
[13] Uwe Wolter, Zinovy Diskin, “The Next Hundred Diagrammatic Specification Techniques, A Gentle Introduction to Generalized Sketches”, 02 September (2015).
[14] Markus Voelter, “DSL Engineering Designing, Implementing and Using Domain-Specific Languages”, http://dslbook.org, (2010 – 2013)
[15] Zeigler, B. P., Sarjoughian, H. S., & Hammonds, P., “Modeling and Simulation-Based Systems Engineering Handbook”. Springer, (2020).