ULBS
Universitatea "Lucian Blaga" din Sibiu

Interdisciplinary Doctoral School

PhD field of study: Computers and Information Technology

PhD Thesis

Categorical Mechanisms in Multi-level Modeling Methods

-SUMMARY-

PhD Student:

Daniel-Cristian Crăciunean

PhD Supervisor:

o. Univ.-Prof. Prof.h.c. Dr. Dimitris Karagiannis

SIBIU 2021

CONTENTS

1 Introduction

The main way to master the complexity and heterogeneity of the systems involved in all areas of activity today is modeling. Conceptual modeling methods that involve the representation of static and dynamic entities of systems support communication between users and developers and facilitate efficient mastery of the complexity and heterogeneity of these systems. Thus, concepts such as internet of things, digital twins, smart factories, smart products, cyber-physical systems, are becoming increasingly common concepts in modeling processes. For example, specifying a flexible manufacturing process involves a variety of systems such as robots, manipulators, software components, control elements, and so on. The result of the aggregation of these components is a Cyber-Physical Production Systems (CPPS) [Muller2018].

By conceptual modeling of a system, we understand the process of thinking and reasoning about a system in order to build a model as well as the process of specifying the resulting model. Modeling is a method of knowing and reflecting on a system in order to replace it with a model more accessible to study. The resulting model must provide reasoning, verification, validation, simulation, and executable code generation mechanisms [Fill2013].

The construction of a model also involves specifying it in a formalism so that it can be used in communication with others or to perform experiments. Building a model usually begins with an informal level useful for discussion and confrontation of ideas, then goes through a semi-formal phase and reaches a formal level, i.e., executable. In general, the formalism of specifying a model influences the way a system is perceived and therefore the way it is modeled.

The idea is validated that diagrammatic languages possess important intuitive features and are therefore best suited for modeling phenomena and the relationships between them [Wolter2015]. Although there are many such languages, the diversity of systems that need to be modeled today makes them often too general to specify models in certain areas. Therefore, the implementation of domain-specific modeling languages (DSML) [Fowler2010] has become an integral part of the modeling process.

A Domain-Specific Language (DSL) is a language with syntax and semantics based on abstractions aligned to a particular domain of knowledge in order to allow the optimal specification of the problems in that domain.

In the context of Model-Driven Development (MDD), the DSLs used to specify models are called Domain-Specific Modeling Language (DSML) and play a fundamental role. The models specified in this way are primary artifacts on the basis of which executable or interpretable models can be generated.

MDD's main objective is to move the software development effort from programming to modeling. This involves, among other things, the automatic generation of executable code based on information extracted from models [Rumpe2017]. But the executable code is a precise, complete and unambiguous specification, i.e., a formal specification. Therefore, this desire imposes higher requirements for the accuracy of the model specifications, because an error in the design of a model could be discovered

only after it already produces large cumulative losses. This inevitably leads to the need to build formal models that can be theoretically verified.

To formally specify a model, we need a formal language, i.e., a language endowed with precise and unambiguous syntax and semantics. It is not enough for the specification language to be formal for the models specified with that language to be formal in turn. Rigorous language can be used to make inaccurate specifications, but for the formal specification of models an essential condition is that the language has to be formal [Rumpe2016].

The fundamental concepts on which this work focuses are: multi-level modeling, modeling method and the MM-DSL language designed to specify the concept of modeling method. To these are added some mechanisms from the category theory that serve to formalize the concepts of multi-level modeling and modeling method. This work contributes, among other things, to the compatibility of the mechanisms of category theory with the concepts involved in the modeling activity.

The use of several levels of models in the sense of specifying models in terms of other models has a long history. Multi-level modeling seems to be the best way to address the reduction of model complexity by abstracting and distributing this complexity to multiple levels. This way of modeling is old but it started to be very important with the increasing complexity of modeled systems [Atkinson2001] [Atkinson2008] [Lara2003] [Lara2010].

A successful approach is the architecture introduced by OMG which classifies models into four layers marked with M0, M1, M2 and M3 [OMG_MDA2014]. Another important approach to multi-level modeling is based on the concept of clabject [Atkinson1997], implemented in an instrument called Melanee [Atkinson2016].

In general, for the development of a domain, it is necessary to define some adequate concepts that will concentrate the effort of an important number of researchers that will contribute to the improvement of the respective concept. One such concept is the modeling method introduced by Karagiannis and Kühn [Karagiannis2002].

The concept of modeling method integrates three essential components of a modeling tool, namely the modeling language, algorithms and mechanisms and the modeling procedure. With these components implemented, according to the modeling procedure, any model can be specified using the modeling language and can be validated, analyzed, executed or simulated using algorithms and mechanisms.

An essential condition for a language to be endowed with formal semantics is that the syntax has to be precise and unambiguous. Most of the time the language associated with a modeling method is a graphic language. If in the case of textual languages there is a strong specification mechanism that offers efficient algorithms for analysis and translation, namely context-free generative grammar, in the case of graphic languages things are far from the same. At first glance it would seem that there is no difference between textual and graphic languages other than the lexical notation of atoms. But there is another detail related to the ordering of these atoms in syntactic constructions. If in the case of textual languages each lexical atom can have at most two neighboring atoms, in the case of graphic languages each atom can interact with an unlimited number of atoms. Hence the benefit related to increased expressiveness of these languages but also a big difficulty related to syntactic analysis.

An important and natural direction for specifying graphic languages is to generalize the mechanisms used in the case of textual languages to graphic languages. Thus, the productions of Chomsky

grammar were replaced with graph transformations. The result is a concept as elegant as textual languages with mathematical support, but no sufficiently efficient analysis algorithms have yet been found to be used in practice similarly to Chomsky grammar. For this reason, the syntax of a modeling language is usually specified by a metamodel or using a mathematical notation. We will use in this work the categorical sketch to specify the syntax of the graphic language. We will, however, use graph transformations to specify the behavioral syntax of a model. In this case we do not need a syntactic analysis but only a matching of the behavioral rules in a model that can be done efficiently by linking these graph transformations, at the metamodel level, to a certain type of elements. Thus, we specify behavioral syntax by associating graph transformations with behavioral action signatures.

If the language syntax is formal then a semantic mapping to a domain with well-known semantics assigns an unambiguous meaning to each syntactic construction specified in the modeling language. Both the static dimension and the behavioral dimension of a model must be endowed with semantics. Thus, the semantics of a modeling language have two dimensions, a static dimension and a behavioral dimension.

The static dimension of semantics in turn has two components, namely a component that endows the elements of the language with types and a component that maps the structures of the model to structures with a well-known semantics.

The behavioral dimension of semantics is most often defined by mapping the domain-specific execution semantics to the semantics of some languages with a formal semantics such as Petri Nets or to certain algebraic specifications. In this work we have specified behavioral semantics through the semantics of graph transformations, which can modify the graph structure of the model, associated with behavioral actions that are mathematical functions and can modify the attributes of the model. These two mechanisms can ensure the transition of the model from one state to another.

Notation refers to the graphic elements of the language and that makes graphic languages more intuitive than textual languages. By notation, geometric figures or icons play the role of lexical atoms in textual languages. Although, in general, the number of graphic lexical atoms is smaller than the number of lexical atoms in textual languages, the more complex relationships between them offer a greater expressiveness of the language.

The notation can be static or dynamic. The static notation is a fixed notation for language elements in the sense that it does not change during model execution. In this case the transition of the model from one state to another is reflected only by changing the values of the attributes. The static notation is specific to most modeling languages. Dynamic notation assumes that the modeling language provides the facility to dynamically change the notation, depending on the state of the model or the value of the attributes of a model in a certain state.

A modeling method must provide facilities for analysis, verification, validation, transformation, querying and simulation of models. Behind these facilities are mechanisms and algorithms that are based on universal properties of models in a particular domain. In our approach, category theory offers a series of universal constructions that can be an important source of generic algorithms.

As we have already mentioned, the development of DSMLs becomes an integral part of the modeling process. These DSMLs are, in general, graphical languages adapted to specify models in a particular domain. The first phase of this process is the specification and implementation of the modeling

method over a metamodeling platform. The next step is to begin specifying the models over this specific modeling tool. If we are in a deadlock, meaning that the chosen metamodeling platform does not give us enough support to be able to specify the domain-specific models, we will have to take it from the beginning by creating a specific modeling tool over another metamodeling platform. If this specific modeling tool were independent of the platform, we could avoid this extra work. This is solved by the MM-DSL language that is thought of as an alternative to building a platform-independent Modeling Method. Platform independence is accomplished by translators specific to each metamodeling platform. The MM-DSL language was specified at the University of Vienna [Karagiannis2016] [Visic2016] and the static part of describing the meta-models was implemented. We have implemented the dynamic part of the language, i.e., the one that offers facilities for specifying computational algorithms.

The process of moving from writing code to writing models inevitably leads to the need to build formal models that can be theoretically verified. Category theory provides a language for the precise description of models, a certain type of model thinking called thinking on arrows and a set of tools in accordance with the design models and structural principles in MDE practice.

Category theory offers both, a language, and a set of conceptual tools to efficiently handle models. All mathematical models can be organized according to their structure by specific categories, and such specific categories can in turn be organized into a more general category that leads to the integration of several models.

An important aspect of modeling is building complex functions from a given set of simple functions, using different operations on functions such as composition and repeat composition. Category theory is exactly the right algebra for such constructions.

The categorical models are a direct support for the modeling and precise mathematical classification of the structures involved in the modeling of the systems as well as of the transformations and operations on the models conveyed in the MDE practice. The operations of composing, transforming, instantiating, synchronizing, etc. are executions of the categorical specifications [Diskin2012].

MDE focuses essentially on the correct structuring of models into submodels, which in turn are further structured similarly. This type of modeling of modern systems requires inter-structural relationships and operations that are not offered by classical mathematics. In the approach of MDE, the operations and relations on the mathematical objects, offered by the category theory, are more important than their internal structure. Just as more models can be composed, through certain specific interactions, to form more complex models, similarly a set of categories can be composed, through specific morphisms, to form a more complex category and therefore both are holistic structures.

In this work we specified both the concept of modeling method and that of multi-level modeling in the language of the category. Important parts of this thesis were published in [Craciunean2018] [Craciunean2019] [Craciunean2019A] [Craciunean2019B] [Craciunean2019C] [Craciunean2019D].

1.1 Problem Statement

The modeling process involves specifying a model, which mimics a real system, from a certain point of view, with a certain degree of fidelity and is able to interact with other real systems or other models in a similar way to the real system. For example, the model of a Cyber-Physical Production System

(CPPS) involves its interaction with a variety of real systems, such as robots, manipulators, and software models, such as control components. Also, software models built into a car must interact with the mechanical components of the car or with other software components of the same car or other cars.

Therefore, the modeling process has as a starting point the real world systems and as an end point the executable models of these systems. The transition from real world systems to executable models is a complex process, which involves passing the model through at least three distinct forms: conceptual model, formal model and executable model.

The executable model is specified in a language that can be interpreted or executed by a machine.

The formal model must be specified precisely in a formal language, i.e., a language with precise syntactic rules, with precise semantics and with appropriate rules for processing information from the domain of use. In general, writing a model in a formal language requires a long experience in specifying formal models and a significant effort from the modeler side. Avoiding this shortcoming can be done only by using a language specific to the modeler's domain of expertise.

The conceptual model is built through the contribution of all parties involved in the modeling process and therefore requires a specification understood by all these parties.

For the specification of a conceptual model to be effective, it needs to be compatible with the modeling tool in which the formal specification is made. This compatibility refers to the existence of a morphism between the conceptual model and the formal model. The lack of compatibility between the two models involves several iterations in the design cycle and therefore leads to increased design and development costs.

For the specification of a conceptual model to be efficient and the model to be understood by all participants in the development process, the formal model, accordingly, must benefit from a modeling tool appropriate to the specific modeling domain.

The diversity and heterogeneity of real systems makes the development of modeling tools and domain specific modeling languages, appropriate, to become a central problem of modeling.

The modeling language must allow, with minimal effort, the precise and intuitive specification of concepts and models in the domain of modeling. In order to fulfill this desideratum such a formal language must have certain natural techniques of modeling and analysis specific to the domain and as a result it will have a limited domain of applicability.

It is obvious the advantage brought by the domain specific modeling tools endowed with domain specific modeling languages (DSML), which bring an efficiency leap comparable to the efficiency leap brought, in programming, by the transition from assembly languages to high-level languages. This advantage derives, in particular, from the fact that a domain specific modeling language is designed for modelers in the domain, not for machines or programmers. But building domain specific tools involves costs. Therefore, it is natural that we want to build an efficient, fast and with minimal costs domain specific modeling tool.

Following the research carried out in the literature from the domain of metamodeling mechanisms and technologies that support the formalization and implementation of modeling methods, we found that concerns in this domain are quite rare.

The main emphasis, in the specialized literature, is on the study of existing languages, on the extension of existing languages to cover new needs and on the construction of new languages and not, enough, on the knowledge, mechanisms and tools to facilitate the optimal development of domain specific modeling tools. Concerns about the mechanisms of constructing formal modeling languages specific to the diagrammatic domain for the precise, systematic and incremental specification of complex systems are difficult to find.

Another major problem related to the implementation of modeling tools is the lack of portability between different metamodeling platforms. After choosing a metamodeling tool and implementing the concept of modeling method on this the next step is to start specifying the concrete models with this specific modeling tool. If the chosen metamodeling platform does not provide us with enough support to be able to efficiently specify the models, we will have to start from the beginning by implementing the modeling method concept over another metamodeling platform. If this specific modeling tool were platform independent, we could avoid this additional work.

If we refer to mathematical formalization, the development of modeling tools is a more experimental and intuitive activity, based on surprisingly weak semantic foundations. The formalization in mathematical language of the models makes them become mathematical objects with adequate properties for analysis and verification. The language offered by category theory has an implicit syntax and an implicit semantics and offers generic algorithms that can be implemented at the metamodel level. Although it is the most expressive mathematical language for specifying models, it is rarely used.

Therefore, the importance of developing domain specific modeling tools associated with the lack of satisfactory concerns in the domain, the remarkable expressiveness of categorical mechanisms in specifying models associated with their insufficient use in the mathematical formalization of metamodels, platform dependence in implementing modeling tools, make up the main motivation of this thesis.

In this idea we aim to answer the general research question:

RQ. What are the theoretical and practical mechanisms that ensure the optimal methodological construction of modeling tools?

In order to systematize the answer to this general question, we will break it down into some more precise questions that will emphasize the basic landmarks of this thesis.

RQ1. What are the appropriate mathematical mechanisms for a formalization of models compatible with metamodeling tools?

If the programming effort involves writing classes or program modules in the case of modeling the main problem becomes the composition of these components, in accordance with the complex interactions between the concepts they represent.

The software components that are composed in a model represent real world concepts and therefore are not just nodes in a graph but are encapsulated algebraic structures and the modeling focuses on operations and relationships on structures considered holistic entities.

Classical mathematics provides mechanisms for modeling atomic concepts but does not provide adequate mechanisms for specifying inter-structural relationships and operations and reasoning on

them, which are essential in specifying models. Category theory offers such mechanisms, moreover these mechanisms are the very essence of category theory. Arrow thinking specific to category theory is appropriate to model-specific inter-structural relationships. These arrows cover all types of interactions between model structures from simple m to n relations specific to ER models to complex functional relationships that transfer the properties of one structure in terms of another structure.

RQ2. What are the appropriate mathematical mechanisms for specifying the syntax of diagrammatic languages?

The design and implementation of domain specific modeling tools has nowadays become an integral stage of the modeling process in accordance with the MDE concept. Domain specific modeling languages (DSML) integrated into these modeling tools are in most cases diagrammatic languages because they are more intuitive. If in the case of textual languages we have context free grammars as a standard mechanism for specifying them with powerful algorithms for their lexical and syntactic analysis, in the case of diagrammatic languages things are still in the phase of searching for equivalent mechanisms and algorithms.

A diagrammatic model has two dimensions; a static dimension and a behavioral dimension. Therefore, specifying the syntax of a diagrammatic modeling language involves specifying the syntax of the two dimensions.

We believe that the categorical sketch is an appropriate mechanism for specifying the syntax of the static dimension of a model. A categorical sketch is a tuple consisting of a graph and a set of constraints on the components of the graph.

To define the syntax of the behavioral dimension we introduced the notion of behavioral rule that composes a graph transformation with a behavioral action. Graph transformations specify local transformations of graphs and behavioral actions are functions that change the values of the attributes associated with the graph components of the model.

RQ3. What are the appropriate mathematical mechanisms for specifying the semantics of diagrammatic languages at the metamodel level?

The static semantics of a model will be defined by mapping the attributes associated with the model concepts to value domains as well as by mapping the structure of the model to known structures.

As we have seen, a behavioral rule is an aggregation between a graph transformation and a behavioral action. We defined the behavioral semantics of diagrammatic models by mapping the behavioral rules to algorithms and mechanisms, implemented at the metamodel level, which execute graph transformations and actions. The graph transformation has, in our approach, two functions, namely, locating the area of definition of the behavioral action and modifying the graph structure of the model. In category theory, the graph transformation is obtained by a double pushout.

RQ4. How can the execution and simulation of a model with categorical mechanisms be specified?

We defined the behavioral semantics of the diagrammatic models by behavioral rules defined at the metamodel level. We defined the states of a model as instances of the static model and the transitions as behavioral rules. The set of instances of a static model together with the set of behavioral rules that can be defined between these instances form a category. We called this category: the category of instances and behavioral transformation (CIBT). This category defines the possible evolution of the

model during the simulation or execution. Each instance represents a possible state of the execution process corresponding to the model. Each state is characterized, therefore, by the values of the attributes associated with the components of the model and by the graph structure of the instance in question. The result is a process represented by a transition system that represents the real behavior of the modeled system [Milner2009] [Aalst2011] [Weske2012].

RQ5. How can the design of modeling tools be methodically approached?

One of the main objectives of this thesis is to identify an optimal flow for the design and implementation of modeling tools. This approach involves sequencing the construction phases of the modeling tool in order to distribute the complexity of the development process in relatively easy to manage steps. The development process is supported by the concept of modeling method on the methodical dimension and by a metamodeling tool on the technological dimension. In our approach the central element of the development process is the concept of modeling method introduced by Karagiannis and Kühn [Karagiannis2002].

The concept of modeling method integrates three essential components of a modeling tool, namely the modeling language, the algorithms and the mechanisms and the modeling procedure. With these components implemented, a modeling tool allows the specification of models, from the modeling domain, in the modeling language according to the modeling procedure, and can be validated, analyzed, executed or simulated using algorithms and mechanisms.

RQ6. How can the components of the concept of modeling method be specified in the context of categorical mechanisms?

The main components of the modeling method concept are the modeling language, algorithms and mechanisms and the modeling procedure. The purpose of these components is to allow the efficient specification and manipulation of concepts in the modeling domain. For the successful implementation of an instance of the concept of modeling method it is not enough for these components to be specified informally, but a precise formal specification of them is needed.

The formalization of a language involves the formal specification of the three essential components of the definition of a language, namely syntax, semantic domain and semantic mapping [Rumpe2016]. For the formal specification of these components, we need a formal language. In this work we use the language provided by category theory for the formal specification of a diagrammatic language.

The design of a DSML must include mechanisms for specifying the two dimensions of a model, namely the static dimension and the behavioral dimension. Both dimensions must be specified by aggregating the same atomic concepts that represent the concepts in the modeling domain. The semantics of a model is defined by mapping syntactic constructions to an appropriate semantic domain that gives meaning to these models.

To specify the syntax of the static dimension of visual models we use the categorical sketch, which offers a formal declarative language endowed with an implicit semantics. Therefore, the mechanism for specifying the static dimension of a visual modeling language is also categorical. The constraints associated with a categorical sketch allow the construction of generic algorithms at the metamodel level to verify the syntactic correctness of the models.

The semantics of a static model is defined by mapping the attributes associated with its atomic components to corresponding data domains and by mapping graph structures to structures with well-defined semantics. A static model represents a state for the behavioral dimension.

To specify the syntax of the behavioral dimension of a visual model we use the concept of behavioral rule which is an association between a graph transformation and a behavioral action. The semantics of graph transformations can be defined by a double pushout that can change the graph structure of the static model transforming it into another static model, and the semantics of behavioral actions through functions that calculate the attribute values of the new static model starting from the attribute values of the old static model.

Mechanisms and algorithms are functional components for both the modeling language and the modeling procedure. Generic components are functional resources for all implementations of modeling methods, specific components only for certain implementations, and hybrid components are generic, but can be configured for multiple implementations [Karagiannis2016]. One of the great advantages of categorical modeling is that category theory is an important resource of generic components. Graphs, graph homomorphisms, categories, functors, diagrams, natural transformations, cones, cocones, boundaries, colimits are just some of these constructions with a generic character and which can be implemented as mechanisms or algorithms, independent of any diagrammatic model that uses them.

RQ7. How can we achieve the portability of metamodels from one metamodeling platform to another?

If after implementing the modeling method concept, for some reason, the metamodeling platform needs to be changed, due to the lack of portability between various platforms, we lose all the work for implementation and we have to start from the beginning by implementing the modeling method concept on the new platform. If this specific modeling tool were independent of the platform, we could avoid this extra work. This is solved by the Modeling Method Domain-Specific Language (MM-DSL) that is thought of as an alternative to building a platform-independent Modeling Method. Platform independence is accomplished by translators specific to each metamodeling platform.

The MM-DSL language was designed and specified at the University of Vienna and includes most relevant metamodeling concepts [Visic2016]. The language contains a subset of descriptive instructions that allow the definition of the components of a modeling method and a subset of calculation and flow control instructions that allow the specification of algorithms. The descriptive part of the language was implemented in the paper [Visic2016] and in this work we completed the implementation of the language with the calculation and control part, i.e., the language constructions that allow the specification of algorithms.

RQ8. How can the mechanisms presented in this work be assembled for the implementation of a concrete modeling tool?

Modeling certain aspects of a system's behavior through processes is a common approach in modeling practice [Aalst2011]. To demonstrate how the mechanisms presented in this work can be assembled in the process of implementing a modeling tool we have presented two possible implementations for two modeling tools, both appropriate for specifying processes, but with different degrees of specificity.

For example, we chose a modeling method with a well-known language: Petri Nets and a specific language for modeling manufacturing processes. In both cases we used the language provided by the

category theory for the mathematical specification of the components of the modeling method concept and then we implemented these examples using as support the MM-DSL language with translation on the ADOxx platform.

1.2 Outline and contributions

Section 2 of the work presents the theoretical and conceptual foundations specific to the modeling domain and a brief analysis of the different existing metamodeling technologies. This section is the result of the systematic study of the mechanisms and knowledge in the domain of research and aims to introduce a common understanding of the concepts of metamodeling, conceptual modeling and model-based development. The content of this section represents the theoretical and instrumental support for the identification and motivation of our research topic as well as for the integration of the research results in this research domain.

Section 3 introduces the notions of model and metamodel by combining the concept of static model, which represents a state of the model, with the notion of behavioral transformation and assembles these concepts into a multi-level model.

This section contributes to the concepts of multi-level modeling, multi-level modeling hierarchy, simulation and co-simulation with support from category theory mechanisms.

We have defined a static model as the image of a functor defined on a categorical sketch, called a metamodel, which respects the constraints imposed and which represents a state of the model. In our approach (section 3.3.) a state of a model is represented by a static structure and a set of values of the attributes.

To define the behavior of a model (section 3.4.) we introduced the notion of behavioral transformation that transforms the structure of the model and the attributes within the limit allowed by the attached constraints. These transformation rules act locally, meaning that they modify several local substates according to certain preconditions. A behavioral transformation is a general description of the local changes that can occur in a model and determines its behavior according to the evolution of the modeled system. In principle, a behavioral rule consists in replacing a part of the model with another part of the construction as well as calculating or recalculating the values of some attributes, in compliance with the constraints imposed by the metamodel.

To generically define the notion of behavioral transformation, we introduced the notions of diagrammatic signature of a behavioral rule and of the diagrammatic signature of an action that specifies how to transform the graph elements at a generic level. Also, for the same purpose we introduced the notion of three-diagram which is a mechanism for associating formal graph components, represented by shape graphs, to actual components in the graph of the categorical sketch.

We then assembled all these elements in the concept of a categorical behavioral sketch that defines the behavior of the models at the metamodel level. Based on the behavioral sketch we then defined the model concept of a behavioral sketch that we called a behavioral pattern (section 3.5.).

Obviously, a model of a system involves combining the static model with the behavioral model in the same diagrammatic format in order to allow the modeler to test and analyze a real system. In this way, we have come to define a metamodel by coupling the static metamodel with the behavioral one. From

this it results that a model, in turn, is obtained by coupling the two dimensions specified by the metamodel, namely the static dimension and the behavioral dimension, the first dimension specifies the model states and the second one transitions from one state to another.

In section 3.6. we introduced the notion of metamodel and the notion of multi-level modeling hierarchy. A multi-level modeling hierarchy can be represented by a tree, which has the root represented by the most abstract metamodel, the inner nodes are metamodels, and the leaves are models. Therefore, each model is directly or indirectly characterized by all the metamodels that are on a typing chain. A multi-level modeling hierarchy specifies a hierarchy that contains a family of models related to different levels of abstraction, which have a common root and two dimensions in which they can be located, a static dimension and a behavioral dimension. The static dimension is represented at each level i of a typing chain. Because typing is unique in each hierarchy, each graph, except the root, has exactly one parent graph in the hierarchy. For each element to have a type and the modeling process to be finite, the root graph has a self-defining collection of individual typing assignments.

Also, in section 3.6. we introduced the notion of indirect three-diagram of a given three-diagram TD, which is a three-diagram obtained by composing a three-diagram TD with a matching p. Multi-level modeling thus creates the opportunity for the implementation of the behavior at a high level of abstraction, through generic behavioral transformations defined at the higher levels and their use at the level of the concrete models.

We can see that, in our approach, behavioral signatures can be introduced at different levels of abstraction together with corresponding three-diagrams and are then mapped by indirect three-diagrams to behavioral transformations that propagate to the least abstract level, i.e., to the level of the concrete model. Therefore, the behavioral transformations can be implemented generically, at a high level of abstraction and then applied to the concrete model.

The behavioral transformations defined in this work are a solid basis for the implementation of a generic model execution engine that allows the simulation of models. In our approach the transition from one state to another is made through behavioral transformations. Of course, certain behavioral transformations can be carried out in parallel in the evolution process of the model. Starting from a theoretic result that says that any two behavioral transformations can be composed in one transformation, called E-concurrent transformation that cumulates the application of two sequential transformations in a single transformation [Ehrig2006] [Ehrig2015], we organized the execution process of a model into a category that we have called a category of instances and behavioral transformations that has as objects instances of the static model and as arrows behavioral transformations. The behavioral transformations between two instances of the model are thus represented by the macro-transitions triggered by the state represented by the initial instance.

Because matching in a model is a complex problem involving NP algorithms, we propose to solve this problem by integrating behavioral transformations into some of the objects and arcs of the model. Specifically, we propose the integration of some calls to the procedures associated with generic behavioral transformations. Procedure calls will be made with appropriate parameters obtained by indirecting the three diagrams. These procedures can be implemented as reactive procedures, i.e., they will be activated every time the local state of the model undergoes a transformation.

Starting from the idea that the simulation starts from an initial state, we defined process simulation category (PSC) as a subcategory of the category of instances and behavioral rules (CIBR).

The PSC category allows the study of the behavior of a system based on the traces of the model in the simulation process, which is identified with the paths in the PSC category that have as initial object the initial instance.

Also, in this section a first evaluation is made of the concepts introduced by short relevant examples and a case study.

Section 4 presents the use of categorical mechanisms to specify the concept of modeling method, presentation accompanied by the specification and implementation in ADOxx of an example of modeling method concept. The essential original contribution of section 4 is related to the categorical specification of the modeling method concept.

The structure of a model and its behavior are essential dimensions of a model and each of them has its own syntax and semantics. Although there is a close connection between the structural semantics of a model and its behavioral semantics, they are still different.

The essential condition for a language to allow the definition of a precise semantics is that it has a rigorous syntax, which clearly and unambiguously specifies the permitted syntactic constructions. The categorical sketch is an ideal concept for rigorously specifying the syntax of a language. The categorical sketch couples very well with the graph transformations and forms, in our opinion, a rigorous, expressive and efficient mechanism for specifying the behavior of a model.

The semantics of a model structure involves mapping attributes to their set of values, interpreting the graph structure of the model, i.e., mapping syntactic constructions to well-defined structures such as sequential, repetitive, recursive, join, fork, meetings, etc. and defining operations on attributes in the context of these structures.

Behavioral semantics of a model involves mapping the model to a set of predicates that check the state of the model and actions (functions) that give outputs from the modeled system, change the state of the model, and transform the graph structure of the model by repositioning components or adding or removing components.

This approach provides an important resource of generic components such as: graphs, graph homomorphisms, categories, functors, diagrams, natural transformations, cones, cocones, limits, colimits, etc. which become fundamental concepts in diagrammatic modeling. All these constructions have a generic character and can be implemented as data types, mechanisms or apriori algorithms, independent of any diagrammatic model that uses them.

The presentation is accompanied by the specification and implementation of an example of a modeling method concept.

Section 5 presents the mechanisms for defining and implementing the MM-DSL language.

Our essential contribution in section 5 is the partial implementation of the MM-DSL language. The MM-DSL language was designed and specified at the University of Vienna by Niksa in his PhD thesis [Visic2016] under the guidance of Professor Karagiannis. The language contains a subset of descriptive instructions that allow the definition of the components of a modeling method and a subset of calculation and flow control instructions that allow the specification of algorithms. The descriptive

part of the language was implemented in the work [Visic2016] and in this work we completed the implementation of the language with the calculation and control part, i.e., the language constructions that allow the specification of algorithms.

The basic concepts of the MM-DSL language and the relations between them as well as the concepts specific to an application model are specified by the metamodel grammar. The graphic representation of these concepts is specified by graphical representation grammar. Mechanisms and algorithms specification is an essential feature provided by the algorithm grammar.

To this essential contribution we can add the SMM specification in the MM-DSL language in order to demonstrate the essential facilities offered by the MM-DSL language for the implementation of modeling methods.

In Section 4 we formalized the basic concepts of a modeling method in the language of category theory. We can see that the MM-DSL language offers us all the necessary facilities to be able to specify a categorical modeling method.

The formalization based on the category theory introduced in sections 4 indicates that the MM-DSL language provides all facilities for the specification of the underlying concepts of a modeling method: representing formal objects that can be class extensions that contain attributes, instance of classes that contain attributes, individual elements or sets of elements, functions or sets of functions and events; representing formal functions that can be mathematical functions, relations and arrows in the sense of graph theory.

Therefore, the MM-DSL language supports the complete implementation of a modeling method concept in a domain-specific modeling tool. This tool includes in addition to a modeling language also basic functionality, specific to a class of models, as well as guidelines and constraints for modeling scenarios according to different modeling purposes. In this way a model is not only a visual specification of a model, but also inherits a specific behavior from the metamodel and also allows structural and semantic processing, including reasoning on the structure or specific properties of the model elements.

Section 6 presents the formalization and implementation of two relevant examples of modeling method, using the MM-DSL language.

The essential contribution from section 6 is the specification and implementation of two relevant examples of modeling method, namely one referring to the construction of a modeling tool based on the Petri Nets language and the other based on a DSML for specifying a certain type of manufacturing process. Therefore, we exemplified the facilities offered by the category theory together with the MM-DSL language for formalizing and implementing the concept of modeling method. This section has the role of evaluating and validating the theoretical and conceptual mechanisms introduced or identified in this thesis.

Section 7 concludes the work by summarizing personal contributions and some conclusions.

2 Theoretical and conceptual foundations

This section of the work presents the theoretical and conceptual foundations specific to the modeling domain and a brief analysis of the different existing metamodeling technologies. The content of this section is the result of the systematic study of the mechanisms and knowledge in the domain of research and aims to introduce a common understanding of the concepts of metamodeling, conceptual modeling and model-based development. The content of this section represents the theoretical and instrumental support for the identification and motivation of our research topic as well as for the integration of the research results in this research domain.

3 Categorical multi-level modeling

The diagrammatic specifications are widespread in modeling. At first, the diagrammatic notations were mainly used as communication specifications between modeling experts, software designers and programmers. This approach led to a diagrammatic modeling, in which the semantic meaning of a construction was approximately and confused.

Nowadays diagrammatic modeling is an integral part of concepts such as Model-Driven Engineering (MDE), Model-Driven Development (MDD), or Model-Driven Architecture (MDA), which are different names of a common approach that aims to move the effort from writing code to creating models as primary artifacts in software development and then generating code directly from these models.

In the context of MDE, modeling is based on the conceptual two-dimensionality of real-world systems and therefore a model is naturally represented by graphs. The current objectives of MDE impose a precise and concise formal syntax and semantics of the diagrammatic notations used to specify the models. The representation of the models through strings of symbols flattens their structure and hides the connections between them thus leading to voluminous, rigid and difficult to understand, validate and analyze specifications.

The category theory offers a graph-based approach, which focuses in particular on the relationships between the components of a model. This approach has as its primary mechanism the categorical sketch that offers a universal formalism for defining the syntax and semantics of graphical languages equivalent to the EBNF formalism used for defining the syntax of programming languages. Our approach regarding the use of the categorical sketch as a mechanism for specifying the static dimension of the models is presented in section 3.2.

In our approach a diagrammatic model is specified in the first phase, static, and then its evolution is specified by transformations of the model. Therefore, static models are images of functors defined on categorical sketches which are metamodels. The details of this approach are presented in section 3.3.

Graphs are an intuitive way to represent the states of a system in visual languages. In this context, the behavior of the systems, thus represented, can be modeled by graph transformations that express local changes of the graphs and are therefore very suitable to describe the local transformations of the model

states. Section 3.4. approaches the problem of modeling the behavior of a model through graph transformations.

Section 3.5. introduces the notions of model and metamodel by combining the concept of static model, which represents a state of the model with the notion of behavioral transformation and the concept of behavioral model. Obviously, a model of a system involves the combination of the two models in the same diagrammatic format in order to allow the modeler to test and analyze a real system.

Section 3.6. presents our approach to multi-level modeling as a method of addressing the reduction of model complexity by abstracting and distributing this complexity on multiple levels.

Section 3.7. treats behavioral transformations as a solid basis for the implementation of a generic model execution engine that allows model simulation. The main advantages of the behavioral transformations are given by their generic nature, a property that allows their implementation at a high level of abstraction, and their visual nature.

4 Categorical Modeling Method

Although there are many modeling tools, the diversity and heterogeneity of modeled systems today makes them too general in many cases and therefore does not provide mechanisms to efficiently specify models in certain specific areas. The solution to this problem is the development of new modeling tools that provide all the necessary mechanisms for the efficient specification of models in these domains.

The concept of modeling method [Karagiannis2011] [Karagiannis2016] abstracts the notion of modeling tool, highlights the fundamental components of a modeling tool and concentrates the efforts of a large group of researchers to find the most appropriate methods for implementing such tools with minimal effort. In this section we focus on the use of categorical mechanisms in the process of formalizing and implementing the concept of modeling method in a modeling tool.

The main components of the modeling method concept are the modeling language, algorithms and mechanisms and the modeling procedure. The purpose of these components is to allow the efficient specification and manipulation of concepts in the domain of modeling. We saw in section 3 that diagrammatic models are characterized by two dimensions, a static dimension and a behavioral dimension. Therefore, these components of a modeling method must allow the specification and manipulation of both dimensions of the models at the metamodel level. Category theory provides the appropriate mechanisms for the generic specification of these components. We specify the static dimension of a model, as we saw in section 3, through the categorical sketch and the behavioral dimension through behavioral rules, mathematical concepts that have a formal syntax and semantics.

In the model based on category theory, mechanisms and algorithms are universal constructions. To specify the syntax of a graphical metamodel we used the categorical sketch, which in turn is represented in a graphical language. The sketch is based on a graph and constraints: commutative diagrams, limits and colimits in categories that are universal constructs and therefore are independent of the metamodel. The concept of natural transformation also has elements of universality, such as naturality property, which can be generically implemented. The arrows of the graph $\boldsymbol{G}$ are sketch

operators that can be implemented at the meta-metamodel level, possibly with small adjustments at the metamodel level. Commutative diagrams, cones and cocoons, can be fully implemented at the meta-metamodel level, thus ensuring the syntactic correctness of any metamodel specified by a sketch.

Mechanisms contribute to increasing modeling efficiency by re-using and standardizing software. A mechanism is an independent, self-contained, reusable software entity that implements a variety of interfaces for the relation with the specified model. Thus, a mechanism is an executable code that can be coupled to the code of other components of a model. Algorithms that come with a modeling method, generally, interpret the model constructs based on universally valid principles for implementing their functionality.

In many ways it is important to represent, for the most part, the semantics of the models in the metamodel. If we consider the model based on category theory then the constructions in the category theory can be implemented at the metamodel level as a package of mechanisms and algorithms that will work coherently in all the specified models.

Part of the content of this section was included by the thesis author in [wp32020].

5 MM-DSL specification and implementation

When the modeling process includes the implementation of a DSML, the first step is to choose an appropriate metamodeling tool and then to specify and implement the modeling method on the selected platform. The next step is to begin specifying the processes over this specific modeling tool. If we are in a deadlock, meaning that the chosen metamodeling platform does not give us enough support to be able to specify the specific real-world processes, we will have to take it from the beginning by creating a specific modeling tool over another metamodeling platform. If this specific modeling tool were independent of the platform, we could avoid this extra work. This is solved by the Modeling Method Domain-Specific Language (MM-DSL) that is thought of as an alternative to building a platform-independent Modeling Method. Platform independence is accomplished by translators specific to each metamodeling platform.

The MM-DSL language was designed and specified at the University of Vienna and includes most relevant metamodeling concepts [Visic2015] [Visic2016]. The language contains a subset of descriptive instructions that allow the definition of the components of a modeling method and a subset of calculation and flow control instructions that allow the specification of algorithms. The descriptive part of the language was implemented in the paper [Visic2016] and in this work we completed the implementation of the language with the calculation and control part, i.e., the language constructions that allow the specification of algorithms.

The conceptualization of a modeling method is largely dependent on the metamodeling platform on which it is to be implemented [Fill2013] [Visic2014]. The specification of metamodel-specific artifacts, such as abstract syntax, abstract semantics, specific generic algorithms, etc., is dependent on the platform selected for implementing the modeling method. This dependence is the main reason that determined the development of a DSL, which should standardize the process of conceptualization, implementation and evaluation of a modeling method. MM-DSL offers the facility to conceptualize and implement a modeling method concept independent of the platform on which it will operate. Of

course, this feature is conditioned by the implementation of a translator specific to this platform. In this way the MM-DSL code can be translated and then compiled and executed on different metamodeling platforms. In this section we present the first MM-DSL language translator that translates a modeling method specified in the MM-DSL language to the ADOxx metamodeling platform.

There are many types of mechanisms used for modeling the syntax of a modeling language including: the mathematical theory of formal languages and algebraic mechanisms. The context free language syntax is often defined by the more widespread variants of generative grammars BNF and EBNF. In this section we will briefly present the EBNF specifications of the MM-DSL language, the tools and mechanisms we used in implementing this DSL.

The syntax of a language is, in fact, a mechanism for the symbolic representation of semantics. Therefore, the implementation of DSML semantics can be done through operational mechanisms, i.e., by translating the syntax of the objects specified in DSML into the syntax of a language with a semantics already implemented on a machine [Voelter2010]. We will therefore translate the models specified in the MM-DSL language into objects specified in the ADOScript language.

In this work we used the Xtext framework which contains all the necessary facilities to specify the syntax of MM-DSL language in EBNF language, for lexical analysis, parsing and construction of abstract syntax tree (AST) in Eclipse Modeling Framework (EMF) model format. To provide these features, Xtext includes the ANTLR (ANother Tool for Language Recognition) tool, which implements the LL(*) parsing algorithm. To specify the translator from the abstract syntax of the MM-DSL language to the abstract syntax of the ADOScript language, we use the Xtend language that facilitates the DSL development process. The Eclipse Modeling Framework (EMF) [Dave2009] [Eric2009] is the host of all these tools and also provides the AST model representation format.

Eclipse development is supervised by the Eclipse Foundation, an independent, nonprofit organization composed of more than 100 companies supporting Eclipse. The Eclipse Modeling Project is the basic element for Eclipse-based model development technologies. It is based on EMF, which provides the basic framework for modeling. Other modeling sub-projects are built over EMF, providing various capabilities.

Eclipse Modeling Framework (EMF) is a modeling framework that exploits the features offered by Eclipse (see section 2.4.3) [Dave2009] [Bettini2016] [Eric2009]. EMF uses structured data to represent models, over which other tools can build specific models. The EMF meta-metamodel integrates very well into Eclipse, but can be used without Eclipse. EMF [OMG2011] [OMG2015A] [OMG2002] is based on the MOF metamodel [OMG_MOF] and has nowadays become a standard technology for building models and modeling languages.

Part of the content of this section was published by the author of this thesis in [Craciunean2019A].

6 Two relevant examples

Models, in general, are built in order to model different aspects of the behavior of a system and therefore the modeling tools must be appropriate to capture, efficiently, these aspects. In order for the

modeling process to be efficient, it is necessary that the same modeling language can be used, both in the informal and in the formal phase, i.e., executable. If the modeling tool is not suitable for the modeling domain, most of the times the informal model cannot be expressed for the understanding of the specialists in the modeling domain. Hence the need to build new modeling tools appropriate to the modeling domain in question.

A common approach in modeling practice is to model certain aspects of a system's behavior through processes. A process contains the tasks to be performed and the order in which it is to be done. Thus, we can consider a process as a procedure specific to a particular type of cases. Essentially, a process is built of tasks, subprocesses, and conditions. Each subprocess is again composed of tasks, conditions, and possibly subprocesses. A task is a logical unit of indivisible work, and therefore it is always accomplished entirely or not at all. If an operation in the body of a task does not execute correctly during a task execution, then the state from the beginning of the task will be restored. In this section we present a possible implementation of two modeling tools, both suitable for specifying processes, but with different degrees of specificity.

We used the language provided by the category theory to formalize two examples of modeling method and then we will implement these examples using as support the MM-DSL language with translation on the ADOxx platform as we saw in section 5. We chose to implement a modeling method with a well-known language: Petri Nets, and a specific language for modeling manufacturing processes. The two examples of implemented modeling method are presented in subsection 6.2. and 6.3.

A Petri Net model is characterized by a static dimension and a behavioral dimension. From a syntactic point of view, the static dimension is a graph with two types of nodes, some that represent places and others that represent transitions.

Obviously, not every graph that has nodes of the two types, places and transitions is a Petri Net model. We will therefore have to add a number of constraints to this type of graph such as the fact that the graph is bipartite, connected and weighted. As we saw in section 5, the syntax of the static dimension of a model can be specified by a categorical sketch in which the constraints are imposed by categorical mechanisms.

The semantics of the static dimension of a Petri Net model is given by the graph structure of the network, by the position of the tokens on places which in our implementation will be represented by values of attributes attached to this type of nodes and by the weights of the arcs. represented by the values of some attributes associated with the arcs.

The dynamic dimension of a Petri Net model is given by the transitions that modify the local distribution of the tokens and by the necessary conditions for triggering the transitions. Triggering a transition is done if the preconditions and postconditions are met and affects only the places in the vicinity of the transition. In our approach, we modeled the behavioral dimension of a model, at the metamodel level, through behavioral rules composed of graph transformations and behavioral actions.

Digitization of current manufacturing systems, which are among the most complex and heterogeneous systems, is a major challenge for MDD. Today's competitive conditions impose, imperatively, the increase of efficiency, which can be achieved only by reconfiguration and rapid adaptation to external requirements. Modeling is the main factor in achieving these goals and, therefore, modeling is the main engine for increasing efficiency. But in order for modeling to be truly a factor in increasing

efficiency, it is also necessary for the modeling process to be efficient. The efficiency of the modeling process is largely dependent on the facilities offered by the modeling tools used. Due to the fact that in the modeling activity of the manufacturing processes several specializations are involved, it is necessary that the used modeling language to provide support in all modeling phases, starting with the informal model specification phase and up to the executable model generation phase.

Due to the great diversity of manufacturing processes, existing modeling languages are often too general and do not provide adequate support for specifying all concepts in the domain of modeling from the informal phase to the executable phase.

In order to meet the modeling requirements from the formal phase to the execution phase, the modeling language must offer a readability accessible to all participants in the modeling process and at the same time mechanisms for precise specification of the models. To meet these two requirements, the modeling language must have a high degree of specificity. Petri Nets, for example, allow a precise specification of models because they are based on a strong mathematical formalism, but are too abstract and therefore have limited readability. Although Petri Nets provide mechanisms for the exact specification of concepts and their relationships, their applicability is limited to a class of models related to process dynamics [Karagiannis2016].

In the case of manufacturing processes, static semantics represented by the graph structure of the models and the values of the attributes is as important as the behavioral semantics represented by the behavioral rules.

Under these conditions, the solution is to implement a new modeling tool, endowed with a modeling language specific to the domain, which would provide such facilities.

The first phase of building a modeling language specific to a modeling domain is conceptual modeling. In this phase, the atomic concepts of the modeling domain are established and what are the characteristics involved in the models we want to specify. Also, in this phase it is established what the models we want to represent should look like and what are the operations to which these models are to be subjected. Therefore, this phase involves structuring all the information in the domain of modeling in terms of concept.

In the next phase, it will have to be established, depending on the requirements of the end users, what the modeling instruments we want to build should look like. In this phase we have a benchmark concept, for specifying the components of the modeling tool, namely the concept of modeling method. We will therefore have to specify the components of the modeling method concept, i.e., the modeling language, the modeling procedure and the mechanisms and algorithms. Of course, the chosen metamodeling instrument also has an essential role in this phase. The MM-DSL metamodeling language has the role of offering flexibility in choosing the metamodeling tool.

It is generally accepted that diagrammatic languages are best suited for specifying models in a particular domain because they are intuitive and expressive enough to provide support in all phases of modeling. The suggestive symbols attributed to atomic concepts provide an intuitive image of the models, and the multiple connection of these atomic components into the models provides a sufficient degree of expressiveness for the informal and formal representation of the models.

In our approach, specifying a model involves specifying the two dimensions, namely the static dimension and the behavioral dimension. Both dimensions must be specified by a syntax, and this

syntax must receive the right meaning by mapping to a semantic domain. Obviously, it is preferable for the behavioral dimension to be specified and implemented, in full, if possible, at the metamodel level. Also, the semantics of the static dimension is preferable to be specified formally, at metamodel levels, and the language to be offered to end users only with the syntax necessary to specify the static dimension of the models and with a textually specified semantics in natural language. Our approach aims to achieve these goals. The algorithms implemented at the metamodel level will be integrated in the algorithms and mechanisms component of the modeling method concept.

In this subsection we will briefly present the conceptualization process of a modeling method that we have called Modeling method for a digital manufacturing planner (MM-DiMaP), and which we will implement in a modeling tool that we named Digital manufacturing planning tool (DiMaP). The DiMaP modeling tool will be endowed with a diagrammatic modeling language that we have called DiMaP-DSML and which is a formal language. For the implementation of the DiMaP modeling tool, we used the MM-DSL language. The example in this subsection is inspired by the manufacturing processes of a company that produces Medical Lasers, for which I have developed software for over 10 years. The content of this subsection was partially included by the author of this thesis in the papers [Craciunean2019C] [wp32020].

7 Conclusions and original contributions

The essence of our approach in this thesis is related to the identification and definition of theoretical and practical mechanisms, appropriate to the process of abstraction of a specific domain and optimal specification of solutions in this domain. This approach is motivated by the imperative need to develop domain specific modeling tools in contrast to the timid concerns in this domain and aims to identify the mechanisms involved in the methodological development of these tools.

The development of graphic languages adapted to specify models in a particular domain, called DSMLs, is an integral part of the software modeling process. Designing a new DSML often involves interpreting the nodes and edges of a graph as model-specific objects and then imposing domain-specific constraints and assigning suggestive visual symbols to represent modeled concepts. Our approach is primarily to find the mechanisms that allow the generic implementation, at the metamodel level, of the syntactic constraints and dynamic behavior of the models.

This paper highlights the compatibility between categorical mechanisms and software modeling mechanisms and contributes to reducing the gap between the abstract nature of category theory and modeling. The categorical sketch represents a diagrammatic metamodel of a class of graphical models and provides a powerful mathematical framework for formalizing the syntax of these models at the metamodel level. The constraints in defining the categorical sketch are imposed by the universal constructions from the category theory or by diagram predicate signature. The universal constructions from the category theory provide us with a package of universally valid results that can be implemented independently of any concrete model at the highest degree of abstraction. The diagram predicate signature can also be implemented to a high degree of abstraction.

To define the behavior of a model (section 3.4.) we introduced the notion of behavioral transformation that transforms the structure of the model and the attributes within the limit allowed by the attached

constraints. These transformation rules act locally, meaning that they modify several local substates according to certain preconditions. A behavioral transformation is a general description of the local changes that can occur in a model and determines its behavior according to the evolution of the modeled system. In principle, a behavioral rule consists in replacing a part of the model with another part of the construction as well as calculating or recalculating the values of some attributes, in compliance with the constraints imposed by the metamodel.

To generically define the notion of behavioral transformation, we introduced the notions of diagrammatic signature of a behavioral rule and of the diagrammatic signature of an action that specifies how to transform the graph elements at a generic level. Also, for the same purpose we introduced the notion of three-diagram which is a mechanism for associating formal graph components, represented by shape graphs, to actual components in the graph of the categorical sketch.

We then assembled all these elements in the concept of a categorical behavioral sketch that defines the behavior of the models at the metamodel level. Based on the behavioral sketch we then defined the model concept of a behavioral sketch that we called a behavioral pattern (section 3.5.).

Obviously, a model of a system involves combining the static model with the behavioral model in the same diagrammatic format in order to allow the modeler to test and analyze a real system. In this way, we have come to define a metamodel by coupling the static metamodel with the behavioral one. From this it results that a model, in turn, is obtained by coupling the two dimensions specified by the metamodel, namely the static dimension and the behavioral dimension, the first dimension specifies the model states and the second one transitions from one state to another.

In section 3.6. we introduced the notion of metamodel and the notion of multi-level modeling hierarchy. A multi-level modeling hierarchy can be represented by a tree, which has the root represented by the most abstract metamodel, the inner nodes are metamodels, and the leaves are models. Therefore, each model is directly or indirectly characterized by all the metamodels that are on a typing chain. A multi-level modeling hierarchy specifies a hierarchy that contains a family of models related to different levels of abstraction, which have a common root and two dimensions in which they can be located, a static or syntactic dimension and a behavioral or semantic dimension. The static dimension is represented at each level i of a typing chain. Because typing is unique in each hierarchy, each graph, except the root, has exactly one parent graph in the hierarchy. For each element to have a type and the modeling process to be finite, the root graph has a self-defining collection of individual typing assignments.

Also, in section 3.6. we introduced the notion of indirect three-diagram of a given three-diagram TD, which is a three-diagram obtained by composing a three-diagram TD with a matching p. Multi-level modeling thus creates the opportunity for the implementation of the behavior at a high level of abstraction, through generic behavioral transformations defined at the higher levels and their use at the level of the concrete models.

We can see that, in our approach, behavioral signatures can be introduced at different levels of abstraction together with corresponding three-diagrams and are then mapped by indirect three-diagrams to behavioral transformations that propagate to the least abstract level, i.e., to the level of the concrete model. Therefore, the behavioral transformations can be implemented generically, at a high level of abstraction and then applied to the concrete model.

The behavioral transformations defined in this work are a solid basis for the implementation of a generic model execution engine that allows the simulation of models. In our approach the transition from one state to another is made through behavioral transformations. Of course, certain behavioral transformations can be carried out in parallel in the evolution process of the model. Starting from a theoretic result that says that any two behavioral transformations can be composed in one transformation, called E-concurrent transformation that cumulates the application of two sequential transformations in a single transformation [Ehrig2006] [Ehrig2015], we organized the execution process of a model into a category that we have called a category of instances and behavioral rules that has as objects instances of the static model and as arrows behavioral rules. The behavioral rules between two instances of the model are thus represented by the transitions triggered by the state represented by the initial instance.

The natural process of formalizing the models in section 3 demonstrates the adequacy of the categorical mechanisms to specify the concepts involved in the metamodeling process as well as the inter-structural relations between them. Also, the categorical language allows the faithful specification of the simulation and execution space of a model. The CIBR category has as objects the states of a model and as arrows the transitions of the model from one state to another and therefore represent the admissible paths of evolution of the model. The components of the CIBR category are objects that can be endowed with properties that can collect any information regarding the evolution of the system, such as the duration, frequency, cost and probability of execution of a transition. This information, if stored in a database, can be used to self-regulate or optimize the system with the help of intelligent analysis and learning tools.

The essential original contribution of section 4 is related to the categorical specification of the modeling method concept. The structure of a model and its behavior are essential dimensions of a model and each of them has its own syntax and semantics. Although there is a close connection between the structural semantics of a model and its behavioral semantics, they are still different.

The essential condition for a language to allow the definition of a precise semantics is that it has a rigorous syntax, which clearly and unambiguously specifies the permitted syntactic constructions. The categorical sketch is an ideal concept for rigorously specifying the syntax of a language. The categorical sketch couples very well with the graph transformations and forms, in our opinion, a rigorous, expressive and efficient mechanism for specifying the behavior of a model.

The semantics of a model structure involves mapping attributes to their set of values, interpreting the graph structure of the model, i.e., mapping syntactic constructions to well-defined structures such as sequential, repetitive, recursive, join, fork, meetings, etc. and defining operations on attributes in the context of these structures.

Behavioral semantics of a model involves mapping the model to a set of predicates that check the state of the model and actions (functions) that give outputs from the modeled system, change the state of the model, and transform the graph structure of the model by repositioning components or adding or removing components.

This approach provides an important resource of generic components such as: graphs, graph homomorphisms, categories, functors, diagrams, natural transformations, cones, cocones, limits, colimits, etc. which become fundamental concepts in diagrammatic modeling. All these constructions

have a generic character and can be implemented as data types, mechanisms or apriori algorithms, independent of any diagrammatic model that uses them.

The presentation is accompanied by the specification and implementation of an example of a modeling method concept.

Our essential contribution in section 5 is the partial implementation of the MM-DSL language. The language contains a subset of descriptive instructions that allow the definition of the components of a modeling method and a subset of calculation and flow control instructions that allow the specification of algorithms. The descriptive part of the language was implemented in the work [Visic2016] and in this work we completed the implementation of the language with the calculation and control part, i.e., the language constructions that allow the specification of algorithms.

To this essential contribution we can add the SMM specification in the MM-DSL language in order to demonstrate the essential facilities offered by the MM-DSL language for the implementation of modeling methods. The MM-DSL language allows the specification, independent of the metamodeling platform, of a modeling method. This implementation demonstrates a very good compatibility of the categorical mechanisms with the syntax and semantics of the MM-DSL language.

The formalization based on the category theory introduced in sections 4 indicates that the MM-DSL language provides all facilities for the specification of the underlying concepts of a modeling method.

The main contribution from section 6 is the specification and implementation of two relevant examples of modeling method, namely one referring to the construction of a modeling tool based on the Petri Nets language and the other based on a DSML for specifying a certain type of manufacturing process. Therefore, we exemplified the facilities offered by the category theory together with the MM-DSL language for formalizing and implementing the concept of modeling method.

The process of conceptualizing, formalizing, specifying and implementing the MM-DiMaP modeling method in the context of categorical mechanisms demonstrates the feasibility and efficiency of our methodological approach. At the end of this process, we obtained the DiMaP modeling tool, which was specified in the MM-DSL language and then translated and tested on the ADOxx metamodeling platform. Although this tool is a prototype endowed only with minimal functionalities, it still demonstrates the main advantages offered by the compatibility of a suitable modeling tool with a limited modeling domain.

The DiMaP-DSML language, based on a small number of lexical atoms, offers sufficient facilities for the formal specification of manufacturing cells as well as facilities for analysis and optimization of manufacturing cells. The rich semantic load of language derives, on the one hand, from the unlimited complexity of the connections between lexical atoms, and on the other hand from the inclusion of behavioral semantics at the metamodel level. Therefore, although the DiMaP-DSML language is formal, it offers facilities for formally specifying a model in the modeling domain similar to the facilities offered by informal languages.

8 References

1. [Aalst2011] Wil M.P. van der Aalst: Process Mining Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag Berlin Heidelberg 2011.
2. [Aalst1998] Wil M.P. van der Aalst, Formalization and verification of event-driven process chains (Computing science reports; Vol. 9801). Eindhoven: Technische Universiteit Eindhoven 1998.
3. [Aalst2004] Wil M.P. van der Aalst and K.M. van Hee: Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA, 2004.
4. [Aho2006] Aho, A.V., Lam, M.S., Sethi, R., Ullman J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Pearson Education, Inc, India (2006)
5. [Aldini2010] Alessandro Aldini, Marco Bernardo, Flavio Corradini, A Process Algebraic Approach to Software Architecture Design, Springer-Verlag London Limited 2010
6. [Asperti1991] Andrea Asperti, Giuseppe Longo, Categories Types And Structures, An Introduction to Category Theory for the working computer scientist, Foundations Of Computing Series M.I.T. Press, 1991.
7. [Andy2005] Andy Ju An Wang, Kai Qian, Component-Oriented Programming ISBN: 978-0-471-64446-0, 2005. Copyright 2005 by John Wiley & Sons, Inc.
8. [ANTLR2018] ANTLR.[Online]. Available: http://www.antlr.org/. December 18, 2018. As of 4.7.2.
9. [Atkinson1997] Colin Atkinson. Meta-modelling for distributed object environments. In Enterprise Distributed Object Computing Workshop [1997]. EDOC'97. Proceedings. First International, pages 90{101. IEEE, 1997.
10. [Atkinson2000] Colin Atkinson and Thomas Khuhne. Meta-level independent modelling. In International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming, pages 12{16. Citeseer, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.26.3964&rep=rep1&type=pdf. 45
11. [Atkinson2001] Colin Atkinson and Thomas Khuhne. The essence of multi-level metamodeling. UMLThe Unified Modeling Language. Modeling Languages,Concepts, and Tools, pages 19{33, 2001. URL http://www.springerlink.com/index/ccbgph1thqmx9myn.pdf. 43, 44, 45, 118,220, 236
12. [Atkinson2002]Colin Atkinson and Thomas Khuhne. Rearchitecting the UML infrastructure.ACM Transactions on Modeling and Computer Simulation,12(4):290{321, October 2002. ISSN 10493301. doi: 10.1145/643120.643123. URL http://portal.acm.org/citation.cfm?id=643120. 643123. 34, 41, 45, 236, 242
13. [Atkinson2003]Colin Atkinson and Thomas Khuhne. Model-driven development:A metamodeling foundation. IEEE Software, 20(5):36{41, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231149. URL http://www.computer.org/portal/web/csdl/doi/10.1109/MS.2003.1231149. 34, 45, 220, 236
14. [Atkinson2008] Colin Atkinson and Thomas Khuhne. Reducing accidental complexity in domain models. Software and Systems Modeling, 7(3):345{359, 2008. ISSN 1619-1366. URL http://dx.doi.org/10.1007/s10270-007-0061-0. 190
15. [Atkinson2014] Colin Atkinson, Ralph Gerbig and Thomas Kuhne. Comparing Multi-Level Modeling Approaches, Proceedings of the 1st Workshop on Multi-Level Modelling co-located with the 17th ACM/IEEE International Conference MODELS 2014, CEUR, Vol-1286, 2014, At Valencia, Spain, Volume: 1286

16. [Atkinson2014A] Colin Atkinson, Georg Grossmann, Thomas Kühne, Juan de Lara (Eds.) ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems September 28 – October 3, 2014 , Valencia (Spain), MULTI 2014 – Multi-nivel Modelling Workshop Proceedings
17. [Atkinson2016] Atkinson C., Gerbig R. Flexible Deep Modeling with Melanee. In: Betz S., Reimer U. (eds.) Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband. Modellierung 2016 Vol. 255. Gesellschaft für Informatik, Bonn, pp. 117–122
18. [Atkinson2017] Colin Atkinson and Thomas Kuhne. On Evaluating Multi-Level Modeling, Published in MODELS 2017
19. [AToM3] http://moncs.cs.mcgill.ca/MSDL/research/projects/AToM3.
20. [ADOxx] ADOxx, https://www.adoxx.org
21. [Bettini2016] Bettini Lorenzo, Implementing Domain-Specific Languages with Xtext and Xtend, Published by Packt Publishing Ltd, Second Edition:August 2016.
22. [Bergstra1985] Bergstra & Klop's Algebra of communicating processes with abstraction, Theoretical Computer Science Volume 37, Pages 77-121, 1985
23. [Barr2012] Michael Barr And Charles Wells, Category Theory For Computing Science-Reprints in Theory and Applications of Categories, No. 22, 2012.
24. [Barr2002] Michael Barr Charles Wells. Toposes, Triples and Theories November 2002.
25. [Bak2012] C. Bak and D. Plump, 'Rooted graph programs', Proceedings of the 7th International Workshop on Graph Based Tools (GraBaTs 2012), Electronic Communications of the EASST, vol. 54, 2012. DOI: 10.14279/tuj.eceasst.54.780.
26. [Bork2014] D. Bork, H.G. Fill, Formal Aspects of Enterprise Modeling Methods: A Comparison Framework, In: System Sciences (HICSS), 47th Hawaii International Conference, pp. 3400-3409 (2014)
27. [Bork2018] Dominik Bork and Robert Andrei Buchmann and Dimitris Karagiannis and Moonkun Lee and Elena-Teodora Miron, An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem, Communications of the Association for Information Systems, Vol. 44, pp. 673-697,2018, Copyright by AIS. DOI: https://doi.org/10.17705/1CAIS.04432
28. [Bork2019] D. Bork, R.A. Buchman, D. Karagiannis, M. Lee, E.T. Miron, An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem, Communications of the Association for Information Systems, forthcoming, http://eprints.cs.univie.ac.at/5462/1/CAIS-OMiLAB-final-withFront.pdf (2019)
29. [Bork2020] 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
30. [Borger2012] E. Borger: Approaches to Modeling Business Processes. A Critical Analysis of BPMN, Workow Patterns and YAWL. in: J. SOFTWARE AND SYSTEMS MODELING, Volume 11, Issue 3 (2012), page 305-318, DOI: 10.1007/s10270-011-0214-z. ISSN: 1619-1366 (print version), ISSN: 1619-1374 (electronic version)
31. [Calude2001] Calude Ch. S., Păun Gh., Computing with Cells and Atoms an introduction to quantum, DNA and membrane computing, Taylor & Francis, (2001).
32. [Campbell2018] G. Campbell, 'Algebraic graph transformation: A crash course', Department of Computer Science, University of York, UK, Tech. Rep., 2018. [Online]. Available: https://cdn.gjcampbell.co.uk/2018/Graph-Transformation.pdf.
33. [Campbell2019] 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.

34. [Cesar2006] Cesar Gonzalez-Perez and Brian Henderson-Sellers. A powertypebased metamodelling framework. Software and Systems Modeling, 5 (1):72{90, 2006. ISSN 1619-1366. URL http://dx.doi.org/10.1007/ s10270-005-0099-9. 33, 45
35. [Christos2008] Christos G. Cassandras, Stéphane Lafortune- Introduction to Discrete Event Systems Second Edition- 2008 Springer Science+Business Media, LLC, ISBN-13: 978-0-387-33332-8
36. [Chen2014] Qingfeng Chen, Baoshan Chen, Chengqi Zhang-Intelligent Strategies for Pathway Mining Model and Pattern Identification, Springer International Publishing Switzerland 2014.
37. [Clark2008] Alexander Clark Francois Coste, Laurent Miclet (Eds.)-Grammatical Inference: Algorithms and Applications 9th International Colloquium, ICGI 2008 Saint-Malo, France, September 22-24, 2008 Proceedings- Springer-Verlag Berlin Heidelberg 2008.
38. [Clark2014] Tony Clark, Cesar Gonzalez-Perez2, Brian Henderson-Sellers. A Foundation for Multi-Level Modelling, ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems September 28 – October 3, 2014 , Valencia (Spain)
39. [Craciunean2018] Daniel-Cristian Crăciunean, Dimitris Karagiannis, Categorical modeling method of intelligent WorkFlow, Lecture Notes in Computer Science, 6th International Conference on Mining Intelligence and Knowledge Exploration, MIKE 2018, VOL 11308, ISSN 3029743, DOI: 10.1007/978-3-030-05918-7_11, 2018.
40. [Craciunean2019A] Daniel-Cristian Crăciunean, Daniel Volovici, MM-DSL, support for implementing modelling tools for manufacturing processes, MATEC Web of Conferences, VOL. 290, ISSN 2261-236X, 2019.
41. [Craciunean2019B] Daniel-Cristian Crăciunean, Dimitris Karagiannis, A categorical model of process co-simulation, INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, VOL 10, NR 1, ISSN 2158-107X, DOI: 10.14569/IJACSA.2019.0100355, 2019.
42. [Craciunean2019C] Daniel-Cristian Crăciunean, Categorical Grammars for Processes Modeling, INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, VOL 10, NR 3, ISSN 2158-107X, DOI: 10.14569/IJACSA.2019.0100105, 2019.
43. [Craciunean2019D] Daniel-Cristian Crăciunean, Categorical Modeling Method, proof of concept for the Petri Net language, MODELSWARD 2019 - Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, ISSN 978-989758358-2, 2019.
44. [Craciunean2020] Daniel-Cristian Crăciunean, "Implementing the Behavioral Semantics of Diagrammatic Languages by Co-simulation" International Journal of Advanced Computer Science and Applications(IJACSA), 11(9), 2020. http://dx.doi.org/10.14569/IJACSA.2020.0110964
45. [Dave2009] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks, EMF: Eclipse Modeling Framework, Addison-Wesley, 2009.
46. [Diskin2008] 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, Pages 19-41
47. [Diskin2012] Zinovy Diskin, Tom Maibaum- Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond, ACCAT 2012
48. [Diskin2018] Diskin Z., König H., Lawford M., 2018. Multiple Model Synchronization with Multiary Delta Lenses. In: Russo A., Schürr A. (eds) Fundamental Approaches to Software Engineering. FASE 2018. Lecture Notes in Computer Science, vol 10802. Springer, Cham
49. [Dodds2008] M. Dodds, 'Graph transformation and pointer structures', PhD thesis, Department of Computer Science, University of York, UK, 2008. [Online]. Available: https://www.cs.york.ac.uk/plasma/publications/pdf/DoddsThesis.08.pdf.

50. [Efendioglu2017] Nesat Efendioglu, Robert Woitsch, Wilfrid Utz and Damiano Falcioni, A Product-Service System Proposal for Agile Modelling Method Engineering on Demand: ADOxx.org, Alexander Rossmann, Alfred Zimmermann (eds.): Digital Enterprise Computing 2017, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 199
51. [Egon2012] Egon Börger,Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL, Software & Systems Modeling ISSN 1619-1366 Volume 11 Number 3 Softw Syst Model (2012) 11:305-318 DOI 10.1007/s10270-011-0214-z, Springer.
52. [Eriksson2013] Eriksson, O., Henderson-Sellers, B., A° gerfalk, P.J.: Ontological and linguistic metamodelling revisited: A language use approach. Information & Software Technology 55(12),2099–2124 (2013)
53. [Eric2009] Eric Clayberg Dan Rubel, Eclipse Plug-ins, Third Edition, Addison-Wesley, 2009.
54. [Ehrig2006] H. Ehrig, K. Ehrig, U. Prange and G. Taentzer, Fundamentals of algebraic graph transformation, ser. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006. DOI: 10.1007/3-540-31188-2.
55. [Ehrig2015] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, Frank Hermann, Graph and Model Transformation General Framework and Applications, Springer-Verlag Berlin Heidelberg 2015
56. [Fill2012] Fill, H-G., Redmond, T. and Karagiannis, D.: FDMM: A Formalism for Describing ADOxx Meta Models and Models. ICEIS, 2012.
57. [Fill2013] Fill, H., Karagiannis, D.: On the conceptualization of modelling methods using the ADOxx meta modelling platform. Enterp. Modell. Inf. Syst. Arch.—Int. J. (2013)
58. [Fowler2010] M. Fowler and R. Parsons, Domain Specific Languages, 1st ed. Addison-Wesley Longman, Amsterdam, 2010.
59. [Glabbeek2001] RJ. van Glabbeek - The Linear Time - Branching Time Spectrum I., The Semantics of Concrete, Sequential Processes- Handbook of Process Algebra-Elsevier Science (2001)
60. [Götzinger2016] David Götzinger, Elena-Teodora Miron and Franz Staffel, OMiLAB: An Open Collaborative Environment for Modeling Method Engineering, in vol. Dimitris Karagiannis, Heinrich C. Mayr John Mylopoulos Editors, Domain-Specific Conceptual Modeling Concepts, Methods and Tools, Springer International Publishing Switzerland 2016
61. [github] https://github.com/metaborg/spoofax
62. [Gomes2016] 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).
63. [Hofstede2010] H. M. Hofstede, A., van der Aalst, W., Adams, M., Russell, N.(eds.): Modern Business Process Automation. Springer, Berlin (2010)
64. [Hoare1985] C.A.R. Hoare - Communicating Sequential Processes, Prentice Hall 1985
65. [Henderson2013] Henderson-Sellers, B., Clark, T., Gonzalez-Perez, C.: On the search for a level-agnostic modeling language. In: Proceedings of the 25th International Conference on Advanced Information Systems Engineering. pp. 240–255. CAiSE'13, Springer-Verlag, Berlin, Heidelberg (2013)
66. [Henderson2013] Henderson-Sellers, B., Eriksson, O., Gonzalez-Perez, C., A° gerfalk, P.J.: Ptolemaic Metamodelling?: The Need for a Paradigm Shift, chap. 4, pp. 90–146. IGI Global (2013)
67. [Henderson2011] Henderson-Sellers B., Bridging metamodels and ontologies in software engineering. Journal of Systems and Software, 84(2):301{313, February 2011. ISSN 01641212. doi: 10.1016/j.jss.2010.10.025. URL http://portal.acm.org/citation.cfm?id=1922690.1922987. 33

68. [Henderson2007] Henderson-Sellers B., and Cesar Gonzalez-Perez The Rationale of Powertype-based Metamodelling to Underpin Software Development Methodologies, 2007
69. [Heckel2018] Reiko Heckel, Gabriele Taentzer (Eds.), Graph Transformation, Specifications, and Nets, Springer International Publishing AG, part of Springer Nature 2018
70. [Hrgovcic2013] Hrgovcic, Vedran; Karagiannis, Dimitris;Woitsch, Robert. Conceptual Modeling of the Organisational Aspects for Distributed Applications: The Semantic Lifting Approach, IEEE 37th Annual Computer Software and Applications Conference Workshops-2013.
71. [Hristakiev2018] I. Hristakiev, 'Confluence analysis for a graph programming language', PhD thesis, Department of Computer Science, University of York, UK, 2018. [Online]. Available: https://etheses.whiterose.ac.uk/20255/.
72. [Jaime2014] Jaime Gómez-Ramirez, A New Foundation for Representation in Cognitive and Brain Science Category Theory and the Hippocampus, Springer Science+Business Media Dordrecht 2014.
73. [Johnson1978] S. C. Johnson, "Yacc: Yet Another Compiler-Compiler," 1978.
74. [Junginger2000] Junginger S., Kuhn H., Strobl R., Karagiannis D. (2000) Ein Geschaftsprozessmanagement-Werkzeug der nachsten Generation – ADONIS: Konzeption und Anwendungen (German). In: Wirtschaftsinformatik 42(5),pp. 392–401
75. [jetbrains] jetbrains.com/mps
76. [Kivunja2017] Charles Kivunja, Ahmed Bawa Kuyini.(2017) Understanding and Applying Research Paradigms in Educational Contexts. International Journal of Higher Education Vol. 6, No. 5; 2017.
77. [Kindler2004] Kindler, E.: On the semantics of BPMNs: A Framework for Resolving the Vicious Circle. In: J. Desel and B. Pernici and M. Weske (Hrsg.), Business Process Management, 2nd International Conference, BPM 2004. volume 3080 of Lecture Notes in Computer Science. S. 82–97. Springer Verlag. 2004.
78. [Kindler2006] E. Kindler, V. Rubin, and R. Wagner. Component Tools: Integrating Petri nets with other formal methods. In S. Donatelli and P. S. Thiagarajan, editors, Application and Theory of Petri Nets 2006, 27th International Conference, volume 4024 of LNCS, pages 37-56. Springer, June 2006.
79. [Karagiannis1996] Dimitris Karagiannis, Junginger S., Strobl R.: Introduction to Business Process Management Systems Concepts. In: Scholz-Reiter B., Stickel E. (eds) Business Process Modelling. Springer, Berlin, Heidelberg, 1996
80. [Karagiannis2002] Dimitris Karagiannis; H. Kühn: 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.
81. [Karagiannis2011] Dimitris Karagiannis N. Visic, Next Generation of Modelling Platforms, Perspectives in Business Informatics Research 10th International Conference, BIR 2011 Riga, Latvia, October 6-8, 2011 Proceedings
82. [Karagiannis2014] Dimitris Karagiannis, Evolution of Knowledge Management: From Expert Systems to Innovation 2.0, IAEA International Conference on Human Resource Development for Nuclear, Power Programs: Building and Sustaining Capacity, 12-16 May 2014
83. [Karagiannis2015] Dimitris Karagiannis, "Agile Modeling Method Engineering," in Proceedings of the 19th Panhellenic Conference on Informatics, Athens, Greece, ACM, 2015, pp.5-10.
84. [Karagiannis2016] Dimitris Karagiannis, Heinrich C. Mayr John Mylopoulos Editors, Domain-Specific Conceptual Modeling Concepts, Methods and Tools, Springer International Publishing Switzerland 2016.

85. [Karagiannis2016A] Karagiannis, D., Buchmann, A.: Linked open models: extending linked open data with conceptual model information. Inf. Syst. 56, 174–197 (2016)
86. [Karagiannis2016C] Dimitris Karagiannis, Robert Andrei Buchmann, Patrik Burzynski, Ulrich Reimer and Michael Walch, Fundamental Conceptual Modeling Languages in OMiLAB in vol. Dimitris Karagiannis, Heinrich C. Mayr John Mylopoulos Editors, Domain-Specific Conceptual Modeling Concepts, Methods and Tools, Springer International Publishing Switzerland 2016.
87. [Karagiannis2018] Dimitris Karagiannis, R.A. Buchmann, A Proposal for Deploying Hybrid Knowledge Bases: the ADOxx-to-GraphDB Interoperability Case, In: System Sciences (HICSS), 51st Hawaii International Conference, pp. 4055-4064 (2018)
88. [Karagiannis2019] Dimitris Karagiannis, Patrik Burzynski, Wilfrid Utz, Robert Andrei Buchmann, A Metamodeling Approach to Support the Engineering of Modeling Method Requirements, IEEE 27th International Requirements Engineering Conference (RE) 2019.
89. [Karagiannis2019A] Dimitris Karagiannis, Dominik Bork, Wilfrid Utz, Metamodels as a conceptual structure: some semantical and syntactical operations ,The Art of Structuring, 2019 Springer, Cham
90. [Kühn2003] H. Kühn; F. Bayer; S. Junginger; D. Karagiannis: Enterprise Model Integration. In: Bauk- necht, K.; Tjoa, A M.; Quirchmayr, G. (Hrsg.): Proceedings of the 4th International Con- ference EC-Web 2003 - Dexa 2003, Prague, Czech Republic, September 2003, LNCS 2738, Springer-Verlag, pp. 379-392.
91. [Lara2002] de Lara, Juan, and HansVangheluwe. 2002.AToM3:Atool for multiformalism and meta-modelling. Lecture Notes in Computer Science 2306:174-88.
92. [Lara2003] de Lara Jaramillo, Juan, Hans Vangheluwe, and Manuel Alfonseca Moreno. 2003. Using meta-modelling and graph grammars to create modelling environments. In Electronic notes in theoretical computer science, vol. 72, edited by Paolo Bottoni and Mark Minas. NewYork: Elsevier.
93. [Lara2010] Juan De Lara and Esther Guerra. Deep meta-modelling with metadepth. In Objects, Models, Components, Patterns, pages 1–20. Springer, 2010
94. [Lara2012] Juan de Lara, Esther Guerra, Ruth Cobos, and Jaime Moreno-Llorena. Extending deep meta-modelling for practical model-driven engineering. The Computer Journal, page bxs144, 2012.
95. [Lauer2019] Fabien Lauer, Gérard Bloch-Hybrid System Identification Theory and Algorithms for Learning Switching Models, Springer Nature Switzerland AG 2019
96. [Lesk1975] M. E. Lesk and E. Schmidt, Lex: A lexical analyzer generator. Bell Laboratories Murray Hill, NJ, 1975.
97. [Manifesto] Manifesto for Agile Software Development, http://agilemanifesto.org/
98. [Macias2018] Fernando Macías, Adrian Rutle, Volker Stolz, Roberto Rodriguez-Echeverria, Uwe Wolter, An Approach to Flexible Multi-level Modelling, Enterprise Modelling and Information Systems Architectures Vol. 13, No. 10 (2018). DOI:10.18417/emisa.13.10
99. [Manes1986] Ernest G. Manes, Michael A. Arbib. Algebraic Approaches to program semantics – Springer – Verlag New York Berlin Heidelberg London Paris Tokyo – 1986
100. [Milner1989] R. Milner- Communication and Concurrency, Prentice-Hall, Inc. Upper Saddle River, NJ, USA 1989.
101. [Milner2009] Robin Milner, The Space and Motion of Communicating Agents, Cambridge University Press, 2009. ISBN 978-0-521-73833-0
102. [Mironescu2020] Mironescu I., Crăciunean D.C., Florea A., Bondrea I. (2020) Improving the Training Methods for Designers of Flexible Production Cells in Factories of the Future. In: Camarinha-Matos L.M., Afsarmanesh H., Ortiz A. (eds) Boosting Collaborative Networks 4.0.

PRO-VE 2020. IFIP Advances in Information and Communication Technology, vol 598. Springer, Cham. https://doi.org/10.1007/978-3-030-62412-5_24
103. [MetaEditA] "MetaEdit+ Workbench User's Guide." [Online]. Available: http://www.metacase.com/support/50/manuals/mwb/Mw.html.
104. [MetaEditB] MetaEdit+ User's Manuals, MetaCase, http://www.metacase.com/support/45/manuals/index.html.
105. [metaborg] http://www.metaborg.org/en/latest/
106. [Mosterman2004] P. J. Mosterman and H. Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. SIMULATION: Transactions of the Society for Modeling and Simulation International, 80(9):433–450, 2004. Special Issue: Grand Challenges for Modeling and Simulation.
107. [Muller2018] Daniel Muller, Christin Schumacher, and Felix Zeidler, Intelligent Adaption Process in Cyber-Physical Production Systems, Leveraging Applications of Formal Methods, Verification and Validation Distributed Systems, 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5–9, 2018, Proceedings, Part III, © Springer Nature Switzerland AG 2018.
108. [OASIS2007] OASIS.Web Services Business Process Execution Language Version 2.0 May 2007, http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.doc
109. [OMG2003] OMG, MDA Guide V1.0, http://doc.omg.org/formal/03-05-01 , May 2003.
110. [OMG2011] OMG. OMG MOF 2 XMI Mapping Specification. (August), 2011.
111. [OMG2010] OMG. Object Constraint Language, v2.2. Management, 03(February),2010.
112. [OMG2015] OMG. Unified Modeling Language: http://www.omg.org/spec/UML/2.5, 2015. formal/2015-03-01
113. [OMG2010A] OMG. Object Management Group. Unified Modeling Language, Infrastructure Version 2.3, 2010. URL http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/. 30, 38
114. [OMG2015A] OMG. Meta Object Facility (MOF) Core Specification: http://www.omg.org/spec/MOF/2.5, 2015. formal/2015-06-05
115. [OMG_BPMN2011] OMG. Business Process Model and Notation (BPMN): http://www.omg.org/spec/BPMN/2.0, 2011. formal/2011-01-03
116. [OMG2001] OMG. CWM Partners; "Common Warehouse Metamodel (CWM)Specification", Feb. 2001 http://www.cwmforum.org/spec.htm.
117. [OMG2002] OMG TC; "MOF 2.0 Query/Views/Transformations RFP", 2002. http://cgi.omg.org/cgi-bin/doc?ad/02-04-10.
118. [OMG2003] OMG. Object Management Group: MDA Guide, Version 1.0.1, 12. June 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf, access 30 November 2004.
119. [OMG2012] OMG. CORBA, 2012, www.corba.org.
120. [OMG_MDA2014] Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0 OMG Document ormsc/2014-06-01
121. [OMG_MDA] http://www.omg.org/mda/
122. [OMG_MOF] http://www.omg.org/mof/
123. [Openmodels] http://www.openmodels.at
124. [Plotkin1981] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.
125. [Paun1998] Gh. Paun, Computing with membranes, J. Computer System Sciences,2000), in press, and TUCS Research Report No. 208, November, (http://www.tucs.fi),(1998).
126. [Padua2011] David Padua (Ed.)- Encyclopedia of Parallel Computing, Springer Science+Business Media, LLC 2011.
127. [Parr2007] T. Parr, The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Bookshelf, 2007.

128. [Peffers2007] Peffers, Ken; Tuunanen, Tuure; Rothenberger, Marcus ; Chatterjee, Samir: A Design Science Research Methodology for Information Systems Research. In: Journal of Management Information Systems 24 (2007), No. 3, pp. 45–77. – ISSN 0742–1222
129. [Plump2010] D. Plump, 'Checking graph-transformation systems for confluence', ECEASST, vol. 26, 2010. DOI: 10.14279/tuj.eceasst.26.367.
130. [Plump2011] D. Plump, 'The design of GP2', in Proceedings 10th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2011), ser. Electronic Proceedings in Theoretical Computer Science, vol. 82, 2011, pp. 1–16. DOI: 10.4204/EPTCS.82.1.
131. [Plump2019] D. Plump, 'Computing by graph transformation: 2018/19', Department of Computer Science, University of York, UK, Lecture Slides, 2019.
132. [SIRIUS] https://www.eclipse.org/sirius
133. [Visic2014] N. Višić and D. Karagiannis, "Developing Conceptual Modeling Tools Using a DSL," in Knowledge Science, Engineering and Management, R. Buchmann, C. V. Kifor, and J. Yu, Eds. Springer International Publishing, 2014, pp. 162–173.
134. [Visic2015] Višić, N., Fill, H.-G., Buchmann, R., Karagiannis, D. 2015. A domain-specific language for modelling method definition: from requirements to grammar. In: Rolland, C., Anagnostopoulos, D., Loucopoulos, P., Gonzalez-Perez, C. (eds.), Proceedings of RCIS 2015, IEEE, 286-297, DOI= 10.1109/RCIS.2015.7128889
135. [Visic2016] N. Višić, Language-Oriented Modeling Method Engineering, Doctoral Thesis , University of Vienna, Scientific Advisor: o. Univ.-Prof. Prof.h.c. Dr. Dimitris Karagiannis (2016)
136. [Voelter2010] Markus Voelter, D S L E n g i n e e r i n g Designing, Implementing and Using Domain-Specific Languages, http://dslbook.org, 2010 – 2013
137. [Walters2006] R. F. C. Walters, Categories and ComputerScience, Cambridge Texts in Computer Science ,Edited by D. J. Cooke, Loughborough University, 2006.
138. [Wainer2011] Gabriel A. Wainer, Pieter J. Mosterman- Discrete-Event Modeling And Simulation Theory and Applications , 2011 by Taylor and Francis Group, LLC
139. [Rus2015] Rus, Teodor. Computer-based problem solving process, World Scientific Publishing Co. Pte. Ltd. 5 Toh Tuck Link, Singapore 596224 ISBN 978-9814663731,2015
140. [Rutle2012] Adrian Rutle, Alessandro Rossini, Yngve Lamoa, UweWolter, A formal approach to the specification and transformation of constraints in MDE, The Journal of Logic and Algebraic Programming Volume 81, Issue 4, May 2012, Pages 422-457
141. [Rumpe2016] Bernhard Rumpe, Modeling with UML Language, Concepts, Methods, Springer International Publishing Switzerland 2016.
142. [Rumpe2017] Bernhard Rumpe, Agile Modeling with UML Code Generation, Testing, Refactoring, Springer International Publishing AG 2017.
143. [Rozsnberg1997] G. Rozsnberg (ed), HANDBOOK OF GRAPH GRAMMARS AND COMPUTING BY GRAPH TRANSFORMATION, Copyright 1997 by World Scientific Publishing Co. Re. Ltd.
144. [Srdjan] Srdjan Živković, Marion Murzek, Harald Kühn, Bringing Ontology Awareness into Model Driven Engineering Platforms BOC Information Systems GmbH, Wipplingerstrasse 1,1010 Vienna, Austria
145. [Spivak2014] David I. Spivak, Category Theory for the Sciences The MIT Press Cambridge, Massachusetts London, England ,2014 Massachusetts Institute of Technology
146. [Sprinkle2010] Sprinkle J., Rumpe B., Vangheluwe H., Karsai G. (2010) Metamodelling – State of the Art and Research Challenges In: MBEERTS Giese, H. et al. (ed.) Vol. LNCS 6100 Springer, pp. 57–76
147. [Spoofax2019] Spoofax Documentation Release 2.5.7 MetaBorg Jul 02, 2019

148. [W3C2004] W3C. Recommendation. RDF Vocabulary Description Language1.0: RDF Schema, 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.
149. [W3C2012] W3C OWL 2 Web Ontology Language Document Overview (Second Edition), 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-overview-20121211/
150. [W3C2002] W3C. Web Services Conversation Language (WSCL) 1.0 14 March 2002 http://www.w3.org/TR/2002/NOTE-wscl10-20020314/
151. [W3C2002A] W3C. Web Service Choreography Interface (WSCI) 1.0 W3C 8 August 2002 http://www.w3.org/TR/2002/NOTE-wsci-20020808
152. [Fokkink2007] Wan Fokkink, Introduction to Process Algebra, Computer Science-Monograph (English), Springer-Verlag, 2nd edition April 10, 2007.
153. [FMI2019] Functional Mock-up Interface for Model Exchange and Co-Simulation,Document version: 2.0.1 October 2nd 2019, https://fmi-standard.org/.
154. [INTO-CPS2017] INTO-CPS Tool Chain User Manual, Deliverable Number: D4.3a Version: 1.0 Date: December, 2017 Public Document, http://into-cps.au.dk.
155. [Weske2012] Weske, Mathias. Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer 2012, ISBN 978-3-642-28615-5, pp. I-XV, 1-403.
156. [Winskel1993] Winskel Glynn, Nielsen Mogens, Models for Concurrency, November 1993
157. [Winskel2009] Winskel Glynn, Topics in Concurrency Lecture Notes, April 2009.
158. [Wieczorek2017] Wojciech Wieczorek-Grammatical Inference Algorithms, Routines and Applications, Springer International Publishing AG 2017.
159. [Wolter2007] U. Wolter and Z. Diskin. The Next Hundred Diagrammatic Specification Techniques – An Introduction to Generalized Sketches. Technical Report Report No 358, Department of Informatics, University of Bergen, July 2007.
160. [Wolter2015] 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.
161. [wp32020] https://digifof.eu/deliverables/wp3-fof-designer-innovative-teaching-methods-and-tools
162. [XtextA] Xtext Documentation, https://eclipse.org/Xtext/documentation/
163. [Xtext2014] Xtext – Community Online]. Available: http://www.eclipse.org/Xtext/community.html. [Accessed: 27-Sep-2014].
164. [Xtend] Xtend Documentation , https://eclipse.org/xtend/documentation/
165. [Zivkovic2007] Zivkovic, S.; Kühn, H.; Karagiannis, D.: Facilitate Modelling Using Method Integration: An Approach Using Mappings and Integration Rules. In: Österle, H.; Schelp, J.; Winter, R.; (Eds.): Proceedings of the 15th European Conference on Information Systems (ECIS2007) - "Relevant rigour - Rigorous relevance", St.Gallen, Switzerland. June 2007, pp. 2038-2050.
166. [Zeigler2019] B.P. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation Discrete Event and Iterative System Computational Foundations, Academic Press (2019)