Grant ANR-12-INSE-0011

# ANR INS GEMOC

## D3.1.1 - Hierarchical component metamodel
## Task T3.1

**Version 1.0**

## Authors

| Author | Partner | Role |
|---|---|---|
| Julien DeAntoni | I3S/INRIA AOSTE | Lead author |
| Matias Vara Larsen | I3S/INRIA AOSTE | Lead author |
| Benoit Combemale | IRISA/INRIA Triskell | Contributor |
| Frédéric Mallet | I3S/INRIA AOSTE | Contributor |

# Contents

# Hierarchical component metamodel

## 1. Introduction

### 1.1 Purpose

The increasing complexity of system development leads to an increasing number of stakeholders involved in the development of a same system. As a way to reduce this complexity, the system is often scattered into artifacts and each artifact can be defined by using a Domain Specific Modeling Language (DSML) suitable to the nature of the artifact and the habits of the stakeholder. This leads to heterogeneous modeling and imposes a composition of the heterogeneous models used during the development. The purpose of this document is to base the semantic composition of models using operators. The operators are defined between languages but they are applied between models. The operators allow us to generate automatically the coordination between behavioral models. We illustrate our proposal through the specification of the synchronous product operator between a TFSM language and a fUML language.

### 1.2 Perimeter

In the last years, computer science was overwhelmed by the increasing complexity of systems. This complexity is naturally propagating to the development of such systems, and increase the number of stakeholders involved in the development of a same system. By using multiple Domain Specific Modeling Languages, model driven engineering proposed a way to scatter the problem into multiple concerns were each stakeholder manipulate a language adequate to its concern. In this context, the DSML efficiently specifies the concepts as well as their behaviour within a particular domain [18]. While the number of DSMLs used has gone up, they have shown to be an effective vehicle for the reuse of knowledge and to ease focusing on a specific concern. The benefits provided by the separation of the preoccupations introduced the problem of heterogeneous modeling, i.e., modeling with various DSMLs. Cyberphysical systems, by their different preoccupations are examples of heterogeneous systems. They are usually composed of very different parts where each part has one or several dedicated DSML (e.g., state machine to specify the control and data flow languages to specify sensors acquisition and treatments). From heterogeneous modeling naturally emerges the need to compose the DSMLs (their concepts and their behaviours) in order to obtain the model of the whole system.

### 1.3 Summary

In the present document we first study different approaches for model composition. We differentiate two different composition approaches: the structural composition and the behavioral composition. Secondly, we propose a framework for comparing model composition approaches. We considered in this section only approach that can automatically relaize a composition (escaping approach proposing manually composed simulation of heterogeneous models). Additionnaly, the framework is general enough to compare both structural and behavioral composition by using a fixed vocabulary, which categorize the approaches. In the last section, we propose a first version of a language for the specification of behavioral composition operators. The operators are defined between languages but they are applied between models. We introduce the Composition Operator Language to specify the behavioral composition of two languages. We illustrate our proposal through the specification of the synchronous product operator between a TFSM language and a fUML language.

# 2. Document-dependant structure

## 2.1 Model Composition in Software Engineering

### *2.1.1 Introduction*

Modern systems are more and more complex. A common way to deal with the complexity is to split the problems into smaller problems. In many systems, like for instance embedded and cyber-physical systems, each sub problems can be addressed by a Domain Specific Modeling Languages (DSMLs). A DSML efficiently specifies the domain concepts as well as their behavioral semantics within a particular domain. It allows domain experts to use a language close to their in-mind concepts, being this way more efficient than with a general purpose language [18]. By definition, a single DSML is not suitable to capture all the properties of the whole system. Several models conforming different DSMLs are then developed (possibly independently). The system specification is then heterogeneous because it is specified by different languages. It is consequently mandatory to *compose* these heterogeneous models in order to provide simulation and/or verification of the overall system. A DSML, like any language, is made up with a syntax and a semantic. The syntax specifies the concepts of the language and their relationships, as well as well formed rules. The semantic describes the expected evolution(s) of the model state during its execution. This distinction is important to differentiate two different composition approaches: the structural composition, which are roughly speaking merging different syntaxes and the behavioral composition, which are scheduling different behavioral parts.

This chapter presents the study of several approaches of model composition. The approaches are categorized into two classes: Structural Composition and Behavioral Composition. The study of the approaches allows us to develop a framework to compare them in the next section.

### *2.1.2 Structural Composition Approaches*

Heterogeneous systems often rely on several Domain Specific Languages (DSMLs). A given DSML specifies a syntax and a semantics suitable to a domain. We focus in this section on the approach proposing solutions for the composition of the syntaxes of DSMLs. Amongst these approaches, some of them proposes to compose the syntax of several DSMLs into a new syntax. The composed syntax picks some characteristics of the source languages. This allows the user to model a heterogeneous system using a unified language. Currently, we know about three kind of such language-level syntax composition: merge, interfacing and refinement. The merge composes two languages that share a concept. These concepts are used as ”join points” to stitch the two languages together into a unified whole. Differently, interfacing composes DSMLs that capture distinct but related domains. The DSMLs does not present joint points and the composition requires an interface between the languages. In other cases, one DSML captures in detail a modeling concept that exists only as a black box in a second DSML. The concept defined in the one DSML refines in other in the second DMSL. These techniques are presented in [12] and they are implemented using the GME framework. A similar approach is proposed in Monticore [24], which provides two techniques for language composition: Inheritance and language embedding. These techniques correspond respectively to refinement and interfacing. The Nervelang framework [6] is an specific approach that buils a custom language from features coming from different languages. An specific feature, such as the syntactical aspect of a loop, is encapsulated in a module block. The blocks can be composed together for generating the compiler/interpreter of the resulting language. This approach can be understood as a ”interfacing” between general purpose programming languages.

The technique used to compose two languages depends of how related are the domain than they describe. In [29], there are identified four relationships among DSMLs. The relationships are vaguely described, but help us to understand which technique could be used for composing the DSML. The ”co-exist with" relationship indicates that the DSMLs are loosely coupled. The DSMLs have to be coupled by a well-defined interfaces. The interfacing technique should be used to compose both languages. The "uses" relationship indicates that the DSMLs share a concept. The DSMLs are highly coupled and their domains intersect in some way. These concepts could be used as join points to merge the two languages together. We could not find a related composition technique for the "constraints" and "implements" relationships.

These approaches compose languages syntax to generate a new one. In a complex system several language may be composed. A unified language may be not easy to handle and the re-usability would be limited. Moreover, the developer has to manually specify how the languages are composed. This task could be difficult in a Heterogeneous system. Also, it is not clear how much these approaches are suitable for separation of preoccupation and development a single system by various domain experts that focus on a specific part of the system. While the previous approaches specify composition between languages, there exist approaches whose the composition provides a single model from several heterogeneous models (*i.e.,* models that possibly conform to different DSMLs) [5, 2, 13, 23, 10]. This brings the problem of the composition between models. The specification defines what elements are selected and how they are composed. The composition results in a new model. The new model may conform to the inputs model languages (homogeneous case), or it may conform to a different language.

In [5], the composition of two models is always identified as a *merge*. The framework identifies a set of merge-related operators to compare different approaches of model merging. The merge operator takes two models as inputs and apply a relationship. The relationship specifies how the models are related one to another. The application of a merge operator results in a model. In this work, they suppose the models conform to the same language, but conclude that the merge operator can be generalized to heterogeneous models. A related approach where the composition is specified by using operators is implemented in the framework MoMENT [2]. MoMENT provides a set of generic operators, which are defined independently of the input languages. The operators are specified using the algebraic language Maude[9]. The generic operators has to become language specific to be applied. The user has to provide a module called signature to instantiate an operator. The instantiated operator can be applied between two models to generate a new model. The input models have to conform to the metamodel whose signature has been used for the instantiation. The input models and the composed model are conform to the same language. MoMENT allows the user to customize an operator adding new equations using Maude. The approach purposes the use of QVT transformation to generate the equations. This avoids the user to deal with the MoMENT's underlying algebra.

Differently, [13] proposes two generic operators to compose models automatically: the matching operator and the merging operator. The matching operator specifies which elements in the language can match and how they can match. The matching operator is specific to a modeling language in order to specify which elements describe the same concept. The merge operator defines how the matched elements are merged. The framework is adapted to add composition capabilities to a particular language. In this work the generic framework has been implemented in a tool called Kompose[1]. The tool adds composition capabilities to the Ecore metamodel. The tool is built using the Kermeta language and is integrated in Eclipse as a plug-in.

Similarly, Epsilon [23][2] purposes one language for specifying the matching operator and one for specifying the merging operator. The matching is expressed using the Epsilon Comparison Language[3]. The merging is expressed using the Epsilon Merge Language[4]. Theses languages are rule based oriented. The matching rules enable the user to specify a comparison algorithm to identify pairs of matching elements. The matching elements are merged through the merging rules. The input models can conform to different metamodels, but they have to conform to the same meta-metamodel (ecore). The output model conforms to a metamodel choosen by the user. The output model has to conform to the meta-metamodel (ecore).

Finally, Atlas Model Weaver is an approach where the composition is defined in a *weaving model*. The weaving model captures the relationship between elements from different models. The weaving model conforms to the weaving metamodel. The weaving metamodel defines the kind of relationship that can be modeled. A typical weaving model is made of two families of relationship kinds: matching and composition

[1] http://www.kermeta.org/kompose/
[2] http://www.eclipse.org/epsilon/
[3] http://www.eclipse.org/epsilon/doc/ecl/
[4] http://www.eclipse.org/epsilon/doc/eml/

relationship. The weaving model is translated into an executable transformation. The generated transformation would then compose the input models into the composed model.

Aspect Oriented Approaches [21] propose an asymmetric composition. One model plays the role of base on which, other (partial) models named *aspects* are weaved. The aspect is made up with a *pointcut* and an *advice*. A pointcut specifies the *join points* in the base model, *i.e.,* the place that will be replaced by the advice during the weaving. The advice contains the (partial) model that is to be woven. This approach is for instance implemented in AspectJ[5], MATA [33] and SmartAdapter [27]. In both approaches, an aspect is made up with a pattern matching and a composition specification. In SmartAdapter the composition specification is called composition protocol. The application of the aspect model to the base model is achieved in two phases. First, a match of the pattern has to be found in the base model. Once a match is found, the composition specification defines how elements are combined. Both tools are homogeneous approaches, *i.e.,* they compose models from the same language.

RAM [22] is an aspect oriented approach, which specifies aspect models defining both structure (using class diagrams) and behavior (using sequence diagrams). The weaving of an aspect structure realized by a class diagram merge. The behavioral weaving in RAM is described in the next section whose goal is survey some of the behavioral composition approaches.

### *2.1.3 Behavioral Composition Approaches*

For a DSML to be executable, it must specify the domain concepts but also its behavioral semantics. There are various ways to specify the behavioral semantics of a language ranging from natural language to the use of formal language. Based on this, a behavioral composition approach composes either the behavioral semantics of two (or more) languages or the behavior of two (or more) models.

The majority of the known approaches for behavioral composition are composing models with the same semantics. This is for instance the case of RAM [22], ADORE [28], AOURN [30] or AspectJ. In these case the composition is asymmetric; also they all compose non executable fragments of models into models. All these approaches specify explicitly (with more or less expressiveness) the place where and how the fragments have to be inserted but this specification is made at the model level and not at the language level. We also have to mention BIP [1], which provides a minimalist but expressive and formal algebra to specify the coordination (*i.e.,* the glue) between different automaton. Once again the composition is made at the model level and is homogeneous.

Some symmetric approaches for model composition specified at the language level exist, but are totally hard-coded and language dependent like for instance the synchronous product of automaton.

Semantic anchoring [8] allows to define the semantic of a language. The semantic of a DSML is specified in a concept called Semantic Unit (SU). A SU is represented by a structure (Abstract Data Model) and a behavioral semantic. The approach specifies the SU in the Abstract State Machine Language[6]. Asml specifies both the structural and the semantics of the SU. A given SU is associated to the structure of a DSML through a mapping. The mapping associates each element in the DSML (AS) with an element in the Abstract Data Model of the SU. This approach has been used for heterogeneous language composition. More precisely it enables the user to compose two SU to generate a new one like shown in [7] where a SU-FSM and a SU-SDF are composed to get a new SU called SU-EFSM. The result is then a new SU. In this work the composition is expressed manually using AsmL so that the composition is actually between three AsmL models, resulting in a new SU that can be used in a language definition. The approach is interesting since it enables the creation of a lnguage with a new "dual" semantics and the result is close to structural approach like Monticore for which the behavioral semantics is explicitly dealt with.

Other approaches allow the user to compose different semantics in a predefined way. This is the case of Ptolemy[11] and ModhelX[3]. Theses are actor-oriented approaches where the abstract syntax is fixed. Actors are in a domain who defines the semantic of the syntax. The semantics is given by a *director*, which implements a Model of Computation. If the director is changed, the semantics is changed. Actors are concurrent components, which communicate with each other through interfaces (*i.e.,* ports). Actors can be composed of other actors and leaf actors are executable. The domain defines the communication semantics

[5]http://eclipse.org/aspectj/
[6]http://research.microsoft.com/en-us/projects/asml/

and execution order among actors. Ptolemy has been used successfully to compose hierarchical finite state machines which are embedded within a variety of concurrent models of computations [14, 26]. In this approaches the composition is based on the semantic adaption between Directors at different hierarchical models. While Ptolemy fixes the way that MoCs are combined, Modhel'X allows the user to specify the adaptation by using Java code in specific semantic adaptation interfaces. These tools, while originally based on formal and very interesting approaches for MoC comparison and composition, does not use any of these formal basis. This allows neither the verification nor the analysis of the composition. A recently work consider using CCSL [4] to formally specify the adaption from one MoC to another. This work could provide the means for analyzing the composition.

## 2.2 Classifying composition techniques

### *2.2.1 Introduction*

As highlighted in the previous section, there exist different composition approaches, which use different techniques and provide different possibilities according to their objectives. This variety of approaches makes them difficult to compare. This is however a key point to be able to eventually propose an adequate language for the specification of behavioral composition operators, at the language level. A first framework for comparing structural composition approaches have been proposed in [20]. This framework provides a good basis for comparison but in one hand it does not cover the behavioral composition case and in the other hand the framework lack a simple but precise vocabulary to ease the understanding of differences between the approaches. A framework for comparing Model of Computations (MoC) is presented in [25] (A similar framework is presented in [19]). MoCs have been used as first order entities during heterogeneous model simulation [11, 31, 16]. [25] and [19] are interesting frameworks, which provide formal background for the comparison of two MoCs as well as formal hints defining under which conditions a MoC composition makes sense. However, it does not help comparing the approaches specifying the composition of the MoCs.

We tried in this section to extend the framework proposed in [20] to make it suitable for both structural and behavioral compositions classification. Also, we specified a precise vocabulary to categorize the approaches.

In the next section, we start with a description of the framework itself before to present the classification of existing approaches in this framework.

### *2.2.2 Composition Classification Framework*

The framework consists in the identification of the different concerns that exists in a composition framework (either structural or behavioral). For each concern, we provide a vocabulary to categorize the techniques used by the different state of the art approaches. All composition approaches have two different operations: the act of matching (named ***matching*** in the remainder) and the act of combining the matched entities (named ***combination***). The matching identifies what are the elements that are involved in the composition. More precisely, the act of matching results in a set of set of elements, where all elements in a same set have to be combined together. The combination is the operation that actually realizes the composition of each set of elements from the set of set of elements. The Epsilon Merge Language (EML [23]) makes clear this distinction by using a specific language for the matching (Epsilon Comparison Language) and for the combination (EML itself). This two operations of a composition framework were not sufficient to classify the existing approaches. We identified two concerns for each of these operations.

The matching consist in selecting *What* elements have to be considered by the composition process, and *When* does these elements are matching and must consequently be combined together. More precisely:

**Framework Concern 1.** *The* ***What*** *concern specifies, from a set of models (either representing a language or conforming a language), what are the concepts to be considered by the When concern.*

Usually, the What contains (explicit or implicit) quantifier(s) over elements in one or more models (*e.g.,* $\forall e1 \in L1 \; and \; \forall e2 \in L2$). It can also have (implicit or explicit) filtering over the type of the elements to be matched (*e.g.,* $e1.type = MyConcept$).

**Framework Concern 2.** *The **When** concern specifies the condition between some elements, for which the elements must be grouped together in order to be combined.*

Usually, the When condition is a boolean expression that compare the selected elements properties (*e.g.,* $e1.name = e2.name$). When the condition holds, the elements are put together in a set, specifying that they have to be combined.

While the matching is clearly determined, the combination is more dependent of the kind of composition and its underlying technical aspect. However, we determined that the combination consists in selecting ***Where*** the result of the combination is stored and determining ***How*** to compose the elements.

**Framework Concern 3.** *The **Where** concern specifies the location(s), in one or more models, from the input model or in another model, where the result of the combination is inserted*

The Where location is usually a hint to know the nature of the composition itself. For instance, if the result of the combination is one of the input model, then the composition is asymmetric (*e.g.,* aspect oriented). Another example is when the result is a new metamodel (*i.e.,* a model describing a language), the approach is producing a new language from input languages.

**Framework Concern 4.** *The **How** concern specifies how to combine a set of elements into one or more element(s).*

The How concern is usually considered as the core of the composition approach. This part is really dependent on the kind of composition realized by the approach. For instance, an aspect oriented method weaving an operation call between two other operation calls is very different from a structural composition approach merging the attributes of two elements into a new one.

Because we want to compare very different approaches, it is mandatory to stress a specific aspect of each of these concerns. We chose to focus on the flexibility of the approach, *i.e.,* on the parametrization possibilities offered by the approach to the user. Consequently, we propose to use the following vocabulary:

**hard-coded** In this case the user can not make any choice since the concern is directly encoded in the tool supporting the approach. For instance if we consider the UML package merge, in this case the How concern is hard coded.

**predefined** In this case the user can parametrize the concern by selecting a predefined behavior in a finite and non extensible list. For instance, in AspectJ behavioral weaving, the user can only choose between *before* or *after* for the Where concern.

**user-defined** In this case, there exist a language to define the behavior of the concern. No attention is put on the expressiveness of the language as soon as it allows to specify the behavior of the concern (with a dedicated language).

Note that being the more flexible is not always an advantageous criteria. It usually makes the specification of the composition more tedious for the user. Nevertheless, we think that this (important) drawback can be overcome by a good tooling, which for instance propose on the shelf composition.

### *2.2.3 Classifying Composition Techniques*

In this section, we are systematically classifying approaches for composition. More precisely in one hand we classified all the approaches that were of interest for composition even if not described by the authors as composition approaches (*e.g.,* ESPER). On the other hand some (interesting) approaches often cited as composition approaches have not been classified since they focus on how to simulate/analyze the result of the composition rather than on how to specify the composition itself.

| Approach | Matching | | Combination | | free comment |
|---|---|---|---|---|---|
| | What | When | Where | How | |
| Epsilon EML[23] | user-defined | user-defined | user-defined | user-defined | no behavioral composition. |
| RAM[22] | hard-coded | user-defined ? | user-defined ? | predefined | two kinds of composition. |
| ESPER[7] | ? | ? | ? | ? | |
| CSP | ? | ? | ? | ? | |
| Kompose[8] | user-defined | user-defined | user-defined | user-defined | no behavioral composition |
| AMW[9] | user-defined | user-defined | ? | user-defined | no behavioral composition |
| MATA[33] | hard-coded | user-defined | hard-coded | user-defined | no behavioral composition |
| SmartAdapters[27] | hard-coded | user-defined | hard-coded | user-defined | no behavioral composition |
| MoMENT[2] | predefined | predefined | predefined | predefined | no behavioral composition |

## 2.3 Proposal: Composition Operator

### 2.3.1 Introduction

The first version of this document presents the heterogeneous component model that supports the coordination of heterogeneous models. In this work a component encapsulates an instance of a model conforming to a DSML. The component exposes internal data an events through ports. A component connects with each other using ports and connectors. The connectors defines how two component communicates to each other. The component allows us to compose heterogeneous models. It is possible to create a system by specifying flat and hierarchical compositions of components. These mechanisms seem to be very expressive according to our experiments. The specification of the coordination between models is painful.

We are interesting to define the composition at the language level guided by operators. This is mainly inspired by the product automata operator. The composition is defined between two languages but the application is between models. The application of the operator generates automatically the coordination (Figure 2.1). A similar procedure has been presented in AMW and Moment. Theses tools specify the structural composition at the level of languages. The specification generates an executable transformation. The executable transformation is applied between models to compose structural models. In the behavioral case, the composition is between behavioral models and the result is a coordination glue. This allows the models to execute synchronized.

The operators are specified in the Composition Operator Language. It uses the language's interface to specify the semantic composition of two languages. The interface provides the structure to support the definition of the operator. The operator specification defines what? and when? the elements of both interfaces are related, and How? they are combined. The Composition Operator Language follows the pattern What?, When? and How? presented in chapter 2 for compositional approaches. This allows us to define formally the semantic composition and to compare the approach with the current existing tools.

This chapter presents the composition of behavioral models guided by operators. We first define the elements to specify the operators at the level of languages. We introduce the Composition Operator Language to specify formally the composition of language semantic. The Composition Operator Language is based in the notion of language's interface to specify the operator. We illustrate our proposal through the specification of the synchronous product operator between a TFSM language and a fUML language.

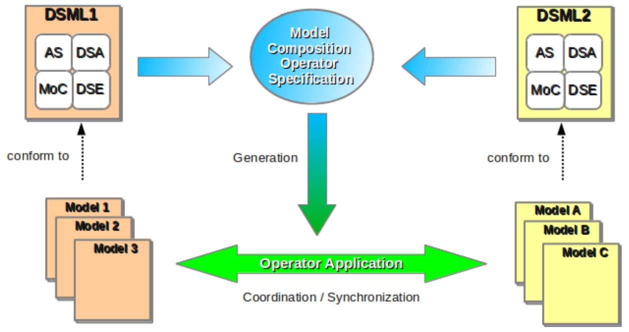

*Figure 2.1: Behavioral model composition workflow*

### 2.3.2 Composition Operator Specification

An operator is defined using the Composition Operator Language (COL). COL uses the language's interface instead of the language itself to express the composition. The interface enables the access to the elements of a DSML. The elements can be in the structure(AS or DSA) or in the behavior (DSE).

We introduce COL specifying the synchronous product operator between a TFSM language and a fUML language. This approach was presented in [32]. The operator specification is shown in Listing 2.1. The top of the specification declares the interface. The interface imports the *TFSM* language and the *fUML* language. The lines 5 and 6 selects *What* elements have to be considered by the composition. In this case, all the pairs of elements (FSMEvent, fUMLEvent) are selected. The line 7 specifies *When* these elements match comparing them by *name*. The matched elements have to be combined together. The line 8 specifies *How* the elements are combined. The combination is expressed through constraints between DSE. The attribute *ocurrs* allows us to get the corresponding DSE. The events are constrained using the relationship *instantaneously implies*. The line 8 can be read as: ”The occurrence of *fsmEvent.occurs implies* the simultaneous occurrence of *fumlEvent.ocurrs*”.

*Listing 2.1: Synchronous product operator*

```
1   import http://TFSM
2   import http://fUML
3
4   operator SynchronousProduct {
5     compose fsmEvent: TFSM.FSMEvent
6     and fumlEvent: fUML.fUMLEvent
7     compare (fsmEvent.name == fumlEvent.name) {
8       (fsmEvent.occurs) instantaneously implies (fumlEvent.occurs)
9     }
10    }
```

Once the operator is specified, it can be applied between models. The application of the operator generates automatically the CCSL specification of the coordination (Listing 2.2). The CCSL relation corresponds with a coincidence (Line 4).

*Listing 2.2: Generated coordination glue in CCSL*

```
1   import TFSM_Model.cssl
2   import fUML_Model.ccsl
3
4   Relation DecisionNode1Coincides_instantaneousExecution [ Coincides ]
5     ( Clock2 -> TFSMEvent1_ocurrs,
6     Clock1 -> fUMLEvent1_ocurrs )
7     }
```

### 2.3.3 Conclusion

We have presented the Composition Operator Language to specify behavioral composition operators at the level of languages. The specification is based in the languages interfaces. The operator defines What? and When? the elements of the interfaces match and How? they are combined. The combination is expressed thought constraints between DSE. The specification is apply between models to generate the coordination. The coordination is a glue expressed in CCSL that enables the synchronization of the models execution.

### 2.3.4 Future Work

Our future research interests have various aspects but we propose here three of them. First, we need to experiment the expressiveness of the proposed approach on various different systems. Second, we have to formally define the Composition Operator Language and the language interface. Third, we also identified that a behavioral composition operator can be characterize by the way it can be used, i.e., by its own properties like for instance associativity, commutativity, etc [15] [17]. Finally, one important characteristic of such operator is its impact on the composition itself. Such characteristics specify the impact of the coordination on the existing properties of each language [15]. Some examples of these characteristics are:

- deadlock-freedom: if model A and model B are deadlock-free, the application of the composition operator results in a deadlock-free behavior.
- safety-secure: if a safety property is true on model A, the application of the composition operator preserves the property.

All such points provide a clear and details road-map for the future works.

## 3. References

[1] Simon Bliudze and Joseph Sifakis. The algebra of connectors: Structuring interaction in bip. In *Proceedings of the 7th ACM &Amp; IEEE International Conference on Embedded Software*, EMSOFT '07, pages 11–20, New York, NY, USA, 2007. ACM.

[2] Artur Boronat, Dr. Jos, Meseguer U. I. Urbana-champaign, Dr. Jos, . Cars, U. P. Valncia, Dr. Reiko, and Heckel U. Leicester. Moment: A formal framework for model management, 2007.

[3] F. Boulanger and C. Hardebolle. Simulation of Multi-Formalism Models with ModHel'X. In *Proceedings of ICST'08*, pages 318–327. IEEE Comp. Soc., 2008.

[4] Frdric Boulanger, Ayman Dogui, Ccile Hardebolle, Christophe Jacquet, Dominique Marcadet, and Iuliana Prodan. Semantic Adaptation Using CCSL Clock Constraints. In *Workshops and Symposia at MODELS 2011*, volume 7167 of *LNCS*, pages 104–118. Springer-Verlag, 2012.

[5] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. A manifesto for model merging. In *Proceedings of the 2006 International Workshop on Global Integrated Model Management*, GaMMa '06, pages 5–12, New York, NY, USA, 2006. ACM.

[6] Walter Cazzola. Domain-specific languages in few steps - the neverlang approach. In Thomas Gschwind, Flavio De Paoli, Volker Gruhn, and Matthias Book, editors, *Software Composition*, volume 7306 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2012.

[7] Kai Chen, J. Sztipanovits, and Sandeep Neema. Compositional specification of behavioral semantics. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[8] Kai Chen, Janos Sztipanovits, and Sandeep Neema. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 35–43, New York, NY, USA, 2005. ACM.

[9] M. Clavel, F. Durn, S. Eker, P. Lincoln, N. Mart-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic, 2001.

[10] Marcos Didonet, Del Fabro, Jean Bzivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *In Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006.

[11] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – the Ptolemy approach. *Proc. of the IEEE*, 91(1):127–144, 2003.

[12] Matthew Emerson and Janos Sztipanovits. Techniques for metamodel composition. In *in The 6th OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2006*, pages 123–139. ACM, ACM Press, 2006.

[13] Franck Fleurey, Benoit Baudry, Robert France, and Sudipto Ghosh. Models in software engineering. chapter A Generic Approach for Automatic Model Composition, pages 7–15. Springer-Verlag, Berlin, Heidelberg, 2008.

[14] A. Girault, Bilung Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(6):742–760, 1999.

[15] Gregor Gössler and Joseph Sifakis. Composition for component-based modeling. In *Formal Methods for Components and Objects*, pages 443–466. Springer, 2003.

[16] Cécile Hardebolle and F Boulanger. Simulation of multi-formalism models with modhelx. In *Proceedings of the first IEEE International Conference on Software Testing Verification and Validation*, 2008.

[17] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. An algebraic view on the semantics of model composition. In *Model Driven Architecture-Foundations and Applications*, pages 99–113. Springer, 2007.

[18] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[19] Axel Jantsch. *Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation*. Systems on Silicon. Morgan Kaufmann Publishers, June 2003.

[20] Cédric Jeanneret, Robert France, and Benoit Baudry. A reference process for model composition. In *Proceedings of the 2008 AOSD Workshop on Aspect-oriented Modeling*, AOM '08, pages 1–6, New York, NY, USA, 2008. ACM.

[21] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akit and Satoshi Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997.

[22] Jacques Klein and Jrg Kienzle. Reusable aspect models. In *IN: Proceedings. Of The 11th International Workshop On Aspect Oriented Modeling.*, 2007.

[23] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. Merging models with the epsilon merging language (eml. In *In Proc. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006*, 2006.

[24] Holger Krahn, Bernhard Rumpe, and Steven Vlkel. Monticore: a framework for compositional development of domain specific languages. *International Journal on Software Tools for Technology Transfer*, 12(5):353–372, 2010.

[25] E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(12):1217–1229, 1998.

[26] Edward A. Lee and Stavros Tripakis. Modal models in ptolemy. In *Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2010)*, pages 1–11, October 2010.

[27] Brice Morin, Gilles Vanwormhoudt, Alban Gaignard, Olivier Barais, and Jean marc Jzquel. Introducing variability into aspect-oriented modeling approaches. In *Vanderbilt University, Springer-Verlag*, pages 498–513, 2007.

[28] Sébastien Mosser and Mireille Blay-Fornarino. "adore", a logical meta-model supporting business process evolution. *Sci. Comput. Program.*, 78(8):1035–1054, 2013.

[29] Sbastien Mosser, Yvan Logre, Nicolas Ferry, and Philippe Collet. From Sensors to Visualization Dashboards: Challenges in Languages Composition. In *GEMOC workshop 2013 - International Workshop on The Globalization of Modeling Languages*, Miami, Florida, USA, September 2013.

[30] Gunter Mussbacher, Daniel Amyot, João Araújo, and Ana Moreira. Requirements modeling with the aspect-oriented user requirements notation (aourn): a case study. In *Transactions on aspect-oriented software development VII*, pages 23–68. Springer, 2010.

[31] Seyed Hosein Attarzadeh Niaki and Ingo Sander. Semi-formal refinement of heterogeneous embedded systems by foreign model integration. In *Specification and Design Languages (FDL), 2011 Forum on*, pages 1–8. IEEE, 2011.

[32] Matias Vara Larsen and Arda Goknil. Railroad Crossing Heterogeneous Model. In *GEMOC workshop 2013 - International Workshop on The Globalization of Modeling Languages*, Miami, Florida, USA, September 2013.

[33] Jon Whittle, Praveen Jayaraman, Ahmed Elkhodary, Ana Moreira, and Joo Arajo. Mata: A unified approach for composing uml aspect models based on graph transformation. In Shmuel Katz, Harold Ossher, Robert France, and Jean-Marc Jzquel, editors, *Transactions on Aspect-Oriented Software Development VI*, volume 5560 of *Lecture Notes in Computer Science*, pages 191–237. Springer Berlin Heidelberg, 2009.