Real-Time Model-Driven Engineering: An Overview

Moussa Amrani and Pierre-Yves Schobbens

Faculty of Computer Science — University of Namur
Rue Grandgagnage, 21
5000 Namur, Belgium
{Moussa.Amrani, Pierre-Yves.Schobbens}@unamur.be

Abstract. After a maturing period of over a decade, Model-Driven Engineering (MDE) starts to extend to novel areas that include safety-critical or embedded, but also cyberphysical systems. These domains explicitly manipulate time. This paper proposes three dimensions to analyse how MDE frameworks integrate time: which transformation paradigm is used, how time is represented inside it, and which characteristics of time are considered. Without claiming for exhaustivity, we validate our approach by analysis several contributions from the literature, thus offering an overview of the current practice in Real-Time MDE.

1 Introduction

After a maturing period of over a decade, Model-Driven Engineering (MDE) has gained enough maturity to extend to novel areas that include safety-critical or embedded, but also cyber-physical systems. These domains make an explicit use of time, forcing language and MDE frameworks to progressively integrate time features to allow the specification of durations, deadlines, delays, etc. Simply borrowing low-level constructs from General-Purpose Programming Languages (GPLs) is not an option any more: this introduces a gap between the high-level concepts captured by models, and the low-level constructs available for expressing time-related computations that become difficult to properly align. Time is not directly accessible to computer: integrating it into MDE frameworks requires to think about which aspects need to be explicitly represented and which could safely be forgotten.

However, one time model could not possibly satisfy all possible needs in MDE. As a first attempt to properly classify Real-Time MDE approaches, this paper proposes a preliminary classification of some of the relevant dimensions, and confronts it against a panel of selected contributions from the literature, thus offering an overview of the current theoretical foundations and practice in the domain. Section 2 explains the classification criteria, and Sections 3 and 4 overview literature contributions based on Meta-Programmed (MPT) and Graph-Based Transformation (GBT) Languages respectively. Section 5 concludes with a summary table relating the analysed contributions to the classification criteria.

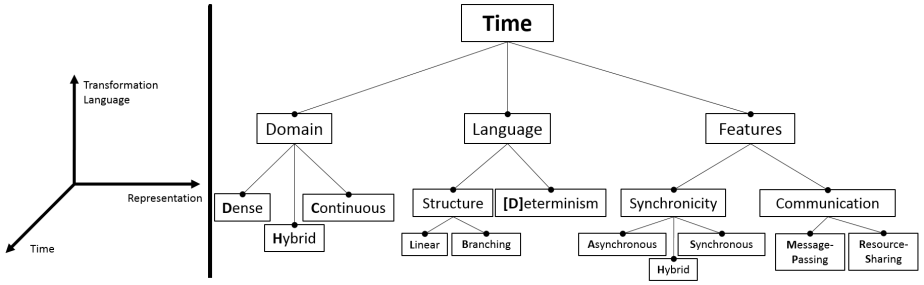

Fig. 1. Left: Three criteria for literature classification: Transformation Language (TL), Time and Representation. **Right:** Details for second criterium: characteristics of Time in Computer Science (adapted from [16]).

2 Model Transformation, Time and Time Representation

This Section explores background material for Model Transformations and Time characteristics and representation to build an analysis grid for classifying the literature contributions (cf. Figure 1, Left).

2.1 Model Transformation: Paradigms and Units

Model Transformation (MT) can be roughly categorised according to their underlying paradigm: *operational* or *meta-programmed* transformations (MPT), adapting constructions from imperative GPLs to model-specific constructions; and *declarative*, relying on rewriting techniques specifying graph-based transformations (GBT) (as a first approximation, see [1] for details). Two elements compose any TL: *transformation units* that bring basic computational bricks expressing a meaningful, well-defined model alteration; and *scheduling mechanisms* orchestrate these units in a more or less explicit fashion. For MPTs, statements constitute the units that are modularly organised into operations, and scheduled implicitly through the statement- and operation call-control flow defined by the TL. Rewriting rules are the basic units for GBTs, and are typically organised in layers, or prioritised; different scheduling mechanisms are possible, ranging from predefined to explicit scheduling DSLs. Time information can appear directly attached to the model, integrated into the transformation units, or as part of the scheduling mechanism.

2.2 Time: Characteristics and Representation

Classical distinctions for real-time systems, such as the difference between *hard*, *soft* and even *firm* real-time (indicating how often and critical deadlines might be missed), or the distinction between *physical* and *logical* time, are not discriminating enough: many systems fall in one or the other depending on the considered viewpoint. This Section attempts to provide a clearer description of the many dimensions of time, by retaining those aspects from Furia, Mandrioli,

Morzenti and Rossi' contribution [16] that seems interesting for providing precise classification criteria.

2.2.1 Characteristics When abstracted for being represented within computers, time is interpreted over a specific *domain*, and manipulated through a *language* that possesses specific *features*: all these characteristics determine how one can reason on a real-time system. The Feature Diagram on the right of Figure 1 summarises these characteristics.

Domain The semantic domain for time consists of a mathematical, numerical set whose intrinsic properties influences the way time can be manipulated by modellers through their language, and the scope of properties they can express. A *dense* set like the rational $\mathbb{Q}$ or real $\mathbb{R}$ numbers allows to find (time) points between any arbitrary pair of points: these sets are perfect for representing "real-life" processes or phenomena in biology, chemistry, physics, etc. On the contrary, a *discrete* set like the integers ($\mathbb{N}$ or $\mathbb{Z}$), allows to specify clock ticks, representing abstractions from the real phenomenon. By using sampling techniques, a dense set can be discretised. A *hybrid* domain makes use of both discrete and continuous time (for different parts in the system). Note that in [16] also appears the notion of *boundedness*: this mostly influences the verification process; besides, none of the overviewed contributions explicitly mentions boundaries.

Language Modellers manipulate the time domain only through their modelling language: it possess a *structure*, exposes a *determinism* model, and allows the specification of a range of properties following a specific *type*.

Structure The states defined by a timed language can be classified into two categories: *linear* formalisms arrange their states into a sequence, implying that state evolution is uniquely determined by the previous state; while *branching* languages organise states as trees where several possible future states may exist concurrently.

Determinism A *deterministic* reactive system can be seen as a system that provides a unique response to an external stimulus in a given state. In several cases however, non-determinism is a powerful abstraction for hiding implementation details, or possible choices on the future states have no obvious reasons (e.g., at design time, it is not relevant to force one task over another), or when the environment (that may produce these inputs) is unsufficiently known (e.g., the chain reaction of a power plant). Stochastic systems bring another precision by associating a probability distribution for specifying undeterministic choices.

Property Type Some languages allow purely *qualitative* constraints, by specifying the relative ordering of some relevant events (e.g., car breaks should activate only after pedal is pushed), without specifying which time actually separate both events; whereas *quantitative* constraints enable a more precise specification (e.g., breaks should activate before 50 ms after pedal push).

Features From a time perspective, it is often practical to specify a system through its components. The language's *features* are the mechanisms available for specifying *synchronicity* and *communication* between these components.

Synchronicity The synchronicity determines how different subsystems, or modules, are paced regarding their relative execution: *synchronous* systems impose changes in modules to occur at the same time, or at times rigidly related; while *asynchronous* systems allow independent progress for each module.

Communication For asynchronous systems (although this theoretically also applies to synchronous ones), a form of communication is needed between modules. Two classical ways are usually used: *message passing*, where communication is achieved by exchanging messages through communication channels; or *resource sharing*, where communication relies on a common resource, usually memory, used to pass information.

Several contributions, especially those targeting heterogeneous systems, actually mix these features together to obtain richer specification, at the expense of complicating the coordination between system modules or languages.

2.2.2 Representation Independently of the time characteristics, what really matters for the modeller is how time is accessible i.e. which MDE elements time information is attached to. We reuse the proposal of [10] as a generalised way of representing time information for all MDE framework (i.e. beyond GBT only).

Time as Data (TaD) , where time information resides inside the model itself, e.g. by integrating clock counters or timers as attributes within specific classes.

Time as Control (TaC) where time information is integrated at the level of transformations, either by integrating it explicitly in the TL, or by extending existing TLs with specific constructions.

Time as Embedding (TaE) where time is not explicitly integrated at the modelling level, but rather implicitly present in an external, third-party language: the TL only offers an implicit represention of time, that only becomes explicit when the whole specification, model(s) + transformation(s), are translated into that external language.

Conceptually however, nothing prevents to mix these representation styles in various ways: indeed, many contributions borrows from two or even three of them. This classification nevertheless provides a good hint on the level of control each contribution offers to modellers, and the following sections classifies the selected contributions by their most prominent approach.

3 Metaprogrammed Transformation Languages

In UML, several diagrams already possess the capability to express time, but at various abstraction levels and in different flavours that are hard to reconcile (cf. Douglass' book for a broad introduction of how time can be handled in the various UML diagrams [14]). Two diagrams attracted the most attention: State Machines (or Statecharts) and Activity Diagrams. In parallel, a recent trend tends to separate the domain concepts, as captured by classical MDE artefacts, from the actual model of computation (MoC): this avoids overloading the model with details pertaining only to the computation; and allows to reason more precisely on the time features.

UML-Based Approaches Because of the many existing variants for Statecharts (von der Beeck mentions 21 variants in [42]), researchers focused on providing a clean semantics of a specific variant [24,43,9], while others contributed with a unified semantic framework integrating several variants ([3,29,26] among others) for reconciling model execution and verification. In parallel, several extensions were explored to overcome existing limitations and addressing variation points: Liu *et al.* proposed a way to integrate explicit communication [25], extending the UML time mechanism that prescribes time values to be solely attached to events; Shankar and Asa integrated time features in a way independent from the underlying diagrams used for model specification [37].

Activity Diagrams serve as a theoretical modelling basis in the Foundational Subset for Executable UML (fUML) [30]. Again, a number of work aimed at clarifying fUML's semantics [23,34]. fUML has been integrated into MOF to operate as a fully-fledged TL, resulting in the xMOF framework [28], served as a semantics specification for UML Profiles in [41], and was leveraged for specifying real-time systems based on fUML concurrency, synchronicity and scheduling model [4].

The MARTE Profile [31] explicitly integrates a multiform notion of time. It can be physical, i.e., continuous or discrete; or logical, i.e. specified through events and clocks whose precise relationships are constrained by means of operators defined in the CCSL (Clock Constraint Specification) Language. No assumption is made *a priori* about the clocks' relative progression or pace. CCSL's semantics is difficult to apprehend, but many contributions targeted a full semantics definition in denotational and operational styles [2,45,12,44]). CCSL has its own simulation tool, TimeSquare [13], which was used by Boulanger, Dogui *et al.* in [7] to define semantic adaptations between different models of computation.

HybridUML [5] is an UML Profile aiming at easing the description of hybrid systems (i.e. systems mixing both discrete and continuous time): it defines new data types for handling both discrete and continuous time domains, and allows to describe systems through agents whose behaviour is specified with hybrid automata that communicate through shared variables.

Separating the MoC from the Model Formal System Design (ForSyDe) (cf. [35,36], and the website `https://forsyde.ict.kth.se/`) proposes a methodology for modelling and designing heterogeneous embedded and cyberphysical systems. ForSyDe focuses on semantic-preserving refinement into executable, low-level implementations obtained from high-level models. The refinement proceeds through controlled transformations, either by refining the design, i.e. the system architecture, by introducing further details; or by preserving the model semantics, by integrating computational details. The methodology enforces determinism in order to perform the proofs guiding the transformation process. The MoC allows either synchronous or asynchronous behaviour. ForSyDe was combined with UML in [21] through the CoMeta methodology [22] to ensure that the intended behaviour of a system is preserved, whatever tool is used to simulate it.

ModHel'X (cf. [18,19] and the website `http://wwwdi.supelec.fr/software/ModHelX`) and GeMoC ([8], cf. the website for further information: `http://gemoc.org/`) are the more advanced frameworks for executing heterogeneous models based on different Models of Computation (MoC). A ModHel'X specification consists of blocks, each with its own MoC, that communicate through well-specified interafaces called pins. The interaction between blocks is handled at the level of interface through adaptors: data is translated back and forth from the outside into the block's computation in a user-defined way. Time constraints imposed by each block propagate through the hierarchical blocks organisation, and may be synchronised through interaction patterns.

GeMoC is more flexible, because many dedicated DSLs allow to fine-tune each aspects of the overall infrastructure: the interaction between the MoC and the internal computation aimed at modifying the model's structure; but also the coordination between heterogeneous models obeying different MoC. However, from an abstract viewpoint, the time model relies on CCSL to specify clock relationships, and on events (corresponding to specific clock ticks) triggering internal model changes. GeMoC is difficult to classify, due to the expressive power of the many DSLs composing its core. By using CCSL, the time domain is clearly heterogeneous, but only qualitative properties can be expressed. Since the communication is also explicitly modelled, both synchronous and asynchronous features can be used, through any of the communication features available.

4 Graph-Based Transformation Languages

Fundamentally, GBT shares with Petri Nets the same algebraic foundations [27]: this provided a good starting point for integrating time into various GBT formalisms, including the possibility to define stochastic transformations. Rewriting logics over algebraic specifications, although similar to Petri Nets, is another distinct source for specifying time within GBT.

Petri Nets-based Approaches Gyapay, Varro and Heckel [17] followed an existing approach for Environment-Relationship High-Level Petri Nets to integrate time into double pushout typed GBT: they introduce dedicated attributes attached to graph vertices to keep track of time elapse; these attributes are updated along the transformation whenever rules are applied. This approach is designed as a quantitative time proposal naturally bound to GBT: it allows to overcome the existing limitations for modelling time, while fully reusing existing tools and technologies for GBT, assuming graphs are modelled from the beginning with time attributes.

Stochastic GBTs were explored by Heckel and his colleagues in [20]: transitions of a Petri Net-like formalism are triggered according to a probability distribution instead of a discrete boolean guard, introducing delays in GBT rules execution. This approach was later enriched with localised events that are ordered through a graph hierarchy, allowing to specify probabilistic dependencies among events residing at different levels. Both contributions target stochastic simulation, although some analysis based on PRISM remains possible.

De Lara and Vangheluwe [11] enrich GBT prioritised rules with time intervals to model imprecise clocks and timeouts, allowing to delay rules execution to a future time specified by the interval. Rules then represent time elapse of

domain-specific actions attached to rules. Again, simulation and analysis are possible through a mapping into Petri Nets, with the results translated back to the original domain-specific concepts.

Rewriting Logics Maude is the common external GPL for Rewriting Logics, thus allowing simulation, reachability analysis and model-checking. Boronat and Ölveczky [6] extended the Moment Framework with Real-Time capabilities by adding timers, clocks and timed values (corresponding to a rated basic clock) to MOF-compliant models. To avoid cluttering MOF-conformant models with time features, they use a kind of "dependency injection", so that model elements concerned with time are referenced by the classes embedding the time information. This approach is a hybrid between TaE/TaD: time information inside the model still has to be explicitly managed by transformations.

The e-Motions framework [33,32] is a visual framework for the modelling, animation and analysis of Real-Time Domain-Specific Languages that provides a large variety of time constructs attached to rules (and therefore, considered as TaC). These constructs define domain-specific actions: rules can be ongoing, meaning that they are constantly updating the model; or atomic with a duration and a period. Such a rich TL requires a careful translation into Maude: atomic rules are translated into two rules, one when a match is found and the other for triggering the model changes; and ongoing rules are managed with low-level primitives in Real-Time Maude.

Other Approaches For the purpose of animating DSLs in (quantitative) real-time even when the model at hand presents several concurrent changes, Strobl and Minas propose another approach based on a DSL aiming at facilitating the specification of animations [38], whose semantics is specified as GBT. For obtaining a reactive system, time information is attached to states, and transitions are triggered by different kinds of events: internal events represent time elapse, making the global time progress; and external events represent user interaction. Events are organised in a queue: the next internal event is computed between all possible events in the graph, while taking into account possible external events that must have priority.

MechatronicUML [15] is a tool suite designed for the modelling and formal analysis of cyber-physical systems using time-dependent dynamic structures that need to be reconfigured architecturally at runtime. Their modelling approach relies on components, whose internal behaviour is specified as real-time statecharts, and on patterns that specify how components communicate without breaking their internal logics. GBT rules specify how the components and their communication ports and channels constituting the overall system architecture, are reconfigured in response to stimuli. The clocks defined in statecharts are accessible to the GBT rules to guard an execution, add, remove or reset clock instances dynamically. By translating the statecharts and the GBT rules as a UPPAAL specification, they are able to perform model-checking.

ATOMPM [40] is a web-based MDE framework relying on a mapping into DEVS [39]: the transformation designer manipulate time directly, but the framework offers concurrency primitives through a dedicated visual language, forcing the designer to handle classical concurrency issues by himself.

	Contribution	Domain	Language	Features	Representation
MPT	Statecharts	**D**	**B**/$\overline{\text{D}}$	**A**/**MP**	**TaC**
	Activity Diags./fUML	**D**	**B**/$\overline{\text{D}}$	**A**/**MP**	**TaC**
	MARTE / CCSL	**H**	**B**/$\overline{\text{D}}$	**H**/**?**	**TaD**+**TaC**
	HybridUML	**H**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaD**+**TaC**
	ForSyDe	**H**	**B**/D	**A**/**RS**	**TaC**
	ModHel'X	**H**	**B**/$\overline{\text{D}}$	**H**/**MP**	TaC
	GeMoC	**H**	**B**/$\overline{\text{D}}$	**H**/**?**	**TaC**
GBT	High-Level Petri Nets	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaC**
	Stochastic Approaches	**H**	**B**/$\overline{\text{D}}$	**A**/**MP**	**TaC**
	De Lara & Vangheluwe	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaC**
	De Lara, Guerra *et al.*	**H**	**B**/$\overline{\text{D}}$	**A**/**MP**	**TaC**+**TaE**
	Strobl & Minas	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaD**
	Gyapay *et al.*	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaD**
	Moment2	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaC**
	e-Motions	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaC**
	MechatronicUML	**H**	**B**/$\overline{\text{D}}$	**A**/**MP**	**TaC**
	AtoMPM	**D**	**B**/$\overline{\text{D}}$	**A**/**RS**	**TaE**

Table 1. Classification of the overviewed contributions. The letters in each column refers to the leafs of the Feature Diagrams in Figure 1(D and $\overline{\text{D}}$ stand for deterministic and undeterministic) and the Representation mode explained in Section 2.2.2.

5 Conclusion

Real-Time MDE is a recent application of MDE techniques and know-hows to new application domains such as embedded, safety-critical and cyber-physical systems for which time is a crucial components. We proposed in this paper a preliminary classification based on three criteria: the transformation language's paradigm (either MPT or GBT); the components characterising the time information available for the modeller; and the way time is represented in the MDE framework. We then overviewed several literature contributions: without aiming at full exhaustivity, the surveyed approaches are in our opinion representative enough for depicting the general picture of the current practice and the recent theoretical developments. Table 1 summarises the results by classifying each (group of) contributions according to our criteria.

We plan to pursue this effort in three directions. First, we plan to collect more contributions than what was possible with this paper's space restriction. Second, we wish to refine our criteria, in particular the Representation: this granularity is not enough to capture adequately the many variations encountered in approaches that handle coordination between models expressed with different MoCs, since this task is complex and happens at different levels. Third, we aim at studying the V&V capabilities of these contributions: functional as well as temporal correctness are crucial features for real-time applications.

Acknowledgments. The authors would like to thank Hans Vangheluwe for pointing us to Furia, Mandrioli, Morzenti and Rossi's book, and James Ortiz for interesting discussions on metaprogrammed frameworks. This research was sponsored by a Ceruna scholarship from the University of Namur, Belgium.

References

1. M. Amrani, L. Lúcio, G. Selim, B. Combemale, J. Dingel, H. Vangheluwe, Y. Le Traon, and J Cordy. Formal Verification Techniques for Model Transformations: A Tridimensional Classification. *JoT*, 2014.
2. C. André. Syntax and Semantics of the Clock Constraint Specification Language. Technical report, Inria, 2009.
3. D. Balasubramanian, C. Pasareanu, M. Whalen, G. Karsai, and M. Lowry. Polyglot: Modeling and Analysis for Multiple Statechart Formalisms. In Issta, 2011.
4. A. Benyahia, A. Cuccuru, S. Taha, F. Terrier, F. Boulanger, and S. Gérard. Extending the Standard Execution Model of UML for Real-Time Systems. In Dipes, 2010.
5. K. Berkenkötter, S. Bisanz, U. Hannemann, and J. Peleska. The HybridUML Profile for UML 2.0. Sttt, 2006.
6. A. Boronat and P. Ölveczky. Formal Real-Time Model Transformations in Moment2. In Fase, pages 29–43, 2010.
7. F. Boulanger, A. Dogui, C. Hardebolle, C. Jacquet, D. Marcadet, and I. Prodan. Semantic Adapatation Using CCSL Clock Constraints. In Mpm *Workshop*, 2012.
8. B. Combemale, C. Brun, J. Champeau, X. Crégut, J. Deantoni, and J. Le Noir. A Tool-Supported Approach for Concurrent Execution of Heterogeneous Models. In Erts, 2016.
9. A. David, M.O. Möller, and W. Yi. Formal Verification of UML Statecharts with Real-Time Extensions. In Fase, 2002.
10. J. De Lara, E. Guerra, A. Boronat, R. Heckel, and P. Torrini. Domain-Specific Discrete Event Modelling and Simulation Using Graph Transformation. SoSyM, 2014.
11. J. De Lara and H. Vangheluwe. Automating the Transformation-Based Analysis of Visual Languages. *Formal Aspects of Computing*, 22(3–4):297–326, 2010.
12. J. Deantoni, C. André, and R. Gascon. CCSL Denotational Semantics. Technical report, Inria, 2014.
13. J. Deantoni and F. Mallet. TimeSquare: Treat your Models with Logical Time. In Tools, 2012.
14. B.P Douglass. *Real Time UML: Advances in the UML for Real-Time Systems*. Addison Wesley, 2004.
15. S. Dziwok, C. Gerking, S. Becker, S. Thiele, C. Heinzemann, and U. Pohlmann. A Tool Suite for the Model-Driven Software Engineering of Cyber-Physical Systems. In Fse *Symposium*, 2014.
16. C. Furia, D. Mandrioli, A. Morzenti, and M. Rossi. *Modeling Time In Computing*. Springer-Verlag, 2012.
17. S. Gyapay, D. Varró, and R. Heckel. Graph Transformation with Time: Causality and Logical Clocks. *Fundamenta Informaticae*, 58(1):1–22, 2003.
18. C. Hardebolle. *Composition de Modèles Pour la Modélisation Multi-Paradigmes*. PhD thesis, University of Paris-Sud XI - Orsay, 2008.
19. C. Hardebolle and F. Boulanger. Multi-Formalism Modelling and Model Execution. *Journal of Computers and their Applications*, 31(3):193–203, 2009.
20. R. Heckel, G. Lajios, and S. Menge. Stochastic Graph Transformation Systems. *Fundamenta Informaticae*, 74(1):63–84, 2006.
21. P. Issa Diallo, J. Champeau, and L. Lagadec. Enhance the Reusability of Models and Their Behavioural Correctness. In GeMoC, 2013.

22. P. Issa Diallo, J. Champeau, and V. Leilde. Model-Based Enginnering for the Support of Models of Computations: the CoMeta Approach. In Mpm, 2011.
23. Q. Lai and A. Carpenter. Static Analysis and Testing of Executable DSL Specification. In *Modelsware*, 2013.
24. D. Latella, I. Majzik, and M. Massink. Towards a Formal Operational Semantics of UML Statechart Diagrams. In Fmoods, 1999.
25. S. Liu, Y. Liu, E. Andre, C. Choppy, J. Sun, B. Wadhwa, and J. Song Dong. A Formal Semantics for Complete UML State Machines with Communications. In *iFM*, 2013.
26. A. Maggiolo-Schettini, A. Peron, and S. Tini. A Comparison of Statecharts Step-Semantics. *Theoretical Computer Science*, 2003.
27. M. Maximova, H. Ehrig, and C. Ermel. Formal Relationship between Petri Net and Graph Transformation Systems Based on Functors between M-Adhesive Categories. In Pn-Gt, volume 40, pages 23–40, 2010.
28. T. Mayerhofer. *Defining Executable Modeling Languages with fUML*. PhD thesis, Technical University of Vienna, 2014.
29. J. Niu, J.M. Atlee, and N.A. Day. Template Semantics for Model-Based Notations. *Transactions on Software Engineering*, 29(10):866–882, 2003.
30. Object Management Group. Semantics of a Foundational Subset for Executable UML Models. Technical report, 2011.
31. Object Management Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Technical report, 2011.
32. J.E. Rivera. *On the Semantics of Real-Time Domain-Specific Modeling Languagues*. PhD thesis, University of Malaga, 2010.
33. J.E. Rivera, F. Durán, and A. Vallecillo. Formal Specification and Analysis of Domain-Specific Models Using Maude. *Simulation*, 85(11–12), 2009.
34. A. Romero, K. Schneider, and M.G. Vieira Ferreira. Using the Base Semantics Given by fUML for Verification. In *Modelsware*, 2014.
35. I. Sander. *System Modeling and Design Refinement in ForSyDe*. PhD thesis, Royal Institute of Technology (Stockholm), 2003.
36. I. Sander and A. Jantsch. System Modeling and Transformational Design Refinement in ForSyDe. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):17–32, 2004.
37. S. Shankar and S. Asa. Formal Semantics of UML with Real-Time Constructs. In Uml, 2003.
38. T. Strobl and M. Minas. Generating Graph Transformation Rules from AML-GT State Machine Diagrams for Building Animated Model Editors. In Agtive, 2011.
39. E. Syriani and H. Vangheluwe. De-/Re-Constructing Model Transformation Languages. In Gt-Vmt *Workshop*, 2010.
40. E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin. AToMPM: A Web-Based Modeling Environment. In *Tool Demo Session — MoDELS Conference*, 2013.
41. J. Tatibouët, A. Cuccuru, S. Gérard, and F. Terrier. Formalizing Execution Semantics of UML Profiles with fUML Models. In MoDELS, 2014.
42. M. von der Beeck. A Comparison of Statecharts Variants. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, 1994.
43. M. von der Beeck. Formalisation of UML Statecharts. In Uml, 2001.
44. M. Zhang and F. Mallet. An Executable Semantics of CCSL and its Applications. In Ftscs, 2015.
45. G. Zholtkevych, F. Mallet, I. Zaretska, and G. Zholtkevych. Two Semantic Models for Clock Relations in the Clock Constraint Specification Language. In Icteri, 2013.