Visual Model-Driven Development of Software Intensive Systems: A Survey of available Techniques and Tools*

Sven Burmester,† Holger Giese, and Stefan Henkler
Software Engineering Group, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany

E-mail: [burmi|hg|shenkler]@uni-paderborn.de

Abstract

Modeling software intensive systems often results in a mix of models from a multitude of disciplines such as software engineering, control engineering, mechanical engineering, and electrical engineering. As software provides the most flexible element in this concert of techniques, the integration of these different views often happens in the software. As today technical systems also become connected to each other using network technology, we no longer only have technical systems which are controlled by an isolated operating software. Instead, the software may include complex information processing capabilities and the coordination between the different technical systems taking hard real-time constraints into account.

In this paper, we at first identify a number of general requirements for the visual model-driven modeling of software intensive systems which result from the outlined demands. We then use these requirements to classify and characterize a large number of today available techniques and tools for software intensive systems. In order to keep this survey focused, we restrict our attention here to techniques and tools for the visual model-driven development of software intensive systems with focus on safety issues and integration of engineering concepts.

1. Introduction

Modeling software intensive systems of different domains is hardly covered by a single notation or family of notations. In addition, often the models from a multitude of disciplines such as software engineering, control engineering, mechanical engineering, and electrical engineering have to be integrated to make up the whole system. Often, this integration is of special relevance for the software, as the software provides the most flexible element in this concert of techniques and thus is the place where the required integration is realized. Especially for technical systems, the software usually has to fulfill safety-critical issues. Further, the software runs either on standard hardware or on embedded controllers. In the latter case, the software has to respect restricted resources. The integration requirements become even more demanding today, as the technical systems are not only controlled by isolated embedded software but also become connected to each other and the outside world using network technology. Besides the coordination of different embedded software elements, this also includes the integration of information system technology or the internet with the embedded software.

To address the outlined challenge with a model-driven development (MDD) approach, visual modeling techniques of the single disciplines such as block diagrams in systems engineering or the Unified Modeling Language (UML) in software engineering, are today employed to foster the design and understanding of complex, software intensive systems. However, a tighter integration between these approaches is needed as the design of future generations of software intensive systems will require a tighter integration between these worlds to really exploit the potential of more ”intelligent” technical systems. Examples for such a trend are self-adaptive [36, 28, 32] and self-optimizing [10] technical systems.

Due to shared roots in the development of the different disciplines, the problem is not as hard as it seems. UML 2.0, for example, already includes concepts such as the components whose origin is the telecommunications domain (cf. ROOM [33] actors). MSC (Message Sequence Charts) and SDL (Specification and Description Language), which also finally found their way into the UML 2.0, are another example. The OMG also acknowledged this demand and therefore published a RFP (Request for Proposal) for *UML*

* This work was developed in the course of the Special Research Initiative 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

† Supported by the International Graduate School of Dynamic Intelligent Systems. University of Paderborn

for Systems Engineering (UML for SE) [29]. The idea of UML for SE is to provide a language that supports the system engineer in modeling and analyzing software, hardware, logical and physical subsystems, data, personnel, procedures, and facilities.

The current integration efforts at the OMG and by commercial vendors between the systems engineering and software engineering domains are characterized by their nature which is often informal. However, many software intensive systems such as transportation, medical applications, command and control systems are safety-critical systems and thus predictability is required. A crucial prerequisite is thus a full semantic integration of the employed visual modeling concepts and their underlying models as otherwise even the clear prediction would not be possible in principle. In addition, support for mathematical analysis of the continuous behavior and for formal methods for the discrete part is required as manual analysis of complex, software intensive analysis seems otherwise infeasible.

In contrast to available surveys which focus on the control perspective and hybrid systems such as [12] or the software/hardware mapping during co-design such as [19], we review in this paper available academic and commercial techniques and tools for their conceptual benefits and shortcomings. Thereby, we regard especially the support for visual model-driven development of software intensive technical systems focussing on software and its potential for "intelligent" technical systems such as self-adaptive [36, 28, 32] or self-optimizing [10] technical systems. In this survey, we evaluate tools with respect to their support for software design beginning with the modeling and resulting in an implementation. It shall guide a user to select a tool that is appropriate for his needs. We do not focus on modeling physical behavior or on specifying requirements for the software. It has to be stressed that the results presented in this survey are only based on the information provided by the cited references and have not been validated by proper experiments with the tools.

The survey is structured as follows: In Section 2, a number of general requirements for any applicable solution to the integration problem for our specific settings are identified. Then, the basic technologies which provide the foundation for most approaches are sketched in Section 3 and the considered techniques and tools are shortly characterized in Section 4. The specific aspect of support for the model-driven development is afterwards discussed in Section 6. The available modeling concepts and their expressiveness including real-time and continuous behavior are then considered in Section 5. In Section 7, the foundations and support for formal analysis for the techniques are reviewed. Finally, we summarize our findings and give an outlook on planned future work.

2. Requirements

To structure the requirements, we will at first look into the requirements for the visual model-driven development, then into the modeling support in general, and finally into the support for formal analysis. The requirements are the taxonomy for the comparison of the approaches (see Table 2).

2.1. Modeling Support

For the modeling support of the approaches, we look in the structural view for means to describe the structure/composition as well as the deployment as both are crucial prerequisites to model complex systems.

As models have to integrate concepts from software, electrical, mechanical, and control engineering, the approaches should support both continuous and discrete behavior. A state machine like notation with possibly hierarchical states, orthogonal states, and history is the criterion for the discrete behavior.

As mentioned in Section 1, future systems will be more intelligent. Such systems require to adapt their behavior and structure to their current context which we call *reconfiguration*. Therefore, we consider whether reconfiguration can be described at all and whether the combination of discrete and continuous behavior is supported by separated units or via embedding as in the case of hybrid automata. In addition, we look if reconfiguration can cross module boundaries and whether the different configurations are exploited to enable an improved resource management.

Obviously, the notion of time is another important issue for modeling real-time systems. Approaches should provide at least one global clock and not only operate with abstract time ticks.

In complex systems, modularity is a major concern a suitable approach has to fulfill. The support for modularity is considered looking whether an appropriate notion of interfaces/modules exist, whether these interface notions are sufficient to ensure correct embedding, and whether the interfaces are static or whether they change during run-time (*dynamic interfaces*).

2.2. MDD Support

A general criterion in the given context is of course that the approach supports the visual description of the system. In addition, we should distinguish whether the approach conforms to a specific standard, is a defacto standard, or a proprietary solution. In addition, often existing standards or proposals are extended.

The levels of the model-driven architecture (MDA) [2] correspond to important steps of MDD. Here, the question is which of the levels platform independent model (PIM),

platform specific model (PSM), and code are populated in the approach. Depending on the supported levels, we can judge whether the tool can be employed in the earlier or later phases only or whether it supports the whole development phases.

An important follow up criterion is then which kind of transformations/synthesis steps are supported and what degree of automation is offered, as this determines to a great extent how well the model-driven philosophy is really supported. We have "PIM → PSM", "PSM → CODE", and the direct synthesis "PIM → CODE". For each of these transformation steps either only the correct timing of the activation is guaranteed or in addition the correct schedulability is included.

2.3. Support for Formal Analysis

Due to the safety-critical character of software intensive systems, formal analyses are required to ensure correctness of the models. As stressed in [12], a formal semantics is a crucial prerequisite for any reasonable modeling approach as otherwise no reasonable analysis of model properties is possible. In addition, such a semantics is required to be implementable (and should be accordingly implemented) as otherwise the analysis result does not necessarily hold for the final system.

A first common approach towards analyzing and understanding the system behavior is usually simulation which derives a possible system trace from the system model. Therefore, we look into the specific support for simulation of the different approaches. As a model is usually mapped to multiple task, the approaches should support schedulability analysis.

To ensure relevant system properties for all possible system traces, we require techniques for the complete formal verification which may be supported in an automatic fashion. Relevant criteria are real-time behavior, real-time reconfiguration, and hybrid behavior. An important requirement in practice is also that a scalable tool support exists which supports modular or compositional reasoning. Therefore, the definition of *refinement* is essential: A model *refines* another one, if it is still compliant with the behavior of the unrefined model.

3. Basis Approaches

In this section, we give an overview about the basis approaches employed in most of the approaches presented in Section 4. Therefore, we first look at the basis approach for continuous and hybrid behavior and then for object-oriented modeling.

Block Diagrams The standard notation for control engineering are block diagrams which are used to specify feedback-controllers as well as the controlled plant. In block diagrams, each block defines a relation between its (continuous) input and output signals (e.g. specified by a differential equation). Arrows connect the input and output signals of the blocks. The standard approach to model an exchange of feedback-controllers is to include discrete blocks. Then, the alternative controller outputs are fed into such a discrete control element whose behavior is described by an automaton. Dependent on the automaton's current discrete state, the according continuous signals are blind out or directed to the block's output. This enables to switch between the output signals of different controllers.

Hybrid Automata A common formal model for hybrid systems are hybrid automata (e.g. [3]) which assign a continuous model (e.g. a feedback-controller) to a discrete state. They include the formalism of timed automata. Hybrid I/O automata [25] further support communicating hybrid automata. Correctness of hybrid automata of small size can be verified by hybrid model checkers (e.g. HyTech,[1] d/dt,[2] RED,[3] CheckMate[4]) .

Hybrid Statecharts [24] defines hybrid statecharts which introduce hierarchic and orthogonal discrete states. This allows modeling more complex systems as these constructs reduce the visual complexity of the models. Their semantics is formally defined which allows formal verification.

ROOM A first object-oriented notation with a technical background was ROOM [33]. It distinguishes between an architectural and a behavioral specification. The architecture of a system is defined in actor diagrams that specify entities called actors, their communication interfaces, and the interconnections used for communication between the actors. The actors' behavior is specified by ROOM charts which are hierarchical statechart models.

UML Today, the defacto standard for software development is the UML. In its newest version, UML 2.0 [30], it provides a large set of different diagram types: E.g., component and classes diagrams to model the structure of a system are as well part of UML as sequence diagrams and state machines for the behavioral specification. Component diagrams in UML 2.0 correspond to the ROOM concept of actor diagrams (see below) and therefore are a first important step of the UML addressing the technical domain.

4. Techniques and Tools

We selected a subset of the available techniques and tools from industry and academia which included MAT-

1 http://www-cad.eecs.berkeley.edu/~tah/HyTech/
2 http://www-verimag.imag.fr/~tdang/ddt.html
3 http://cc.ee.ntu.edu.tw/~val/
4 http://www.ece.cmu.edu/~webk/checkmate

LAB/Simulik/Stateflow, CHARON, Masaccio and Giotto, Hybrid Bond Graphs, HybridUML in combination with HL^3, the triple HyROOM/HyCharts/Hybrid Sequence Charts, MechatronicUML, Ptolemy II, SysML, and Time Weaver. The approaches and the employed references are summarized in Table 1.

MATLAB/Simulink/Stateflow The de facto industry standard employing block diagrams is MATLAB/Simulink and Stateflow. Conditionally blocks, which are only evaluated if explicitly triggered, are an option to model reconfiguration. Thus, a stateflow can be used to only trigger the required elements of the currently active configuration instead of blinding out the results of the ones that are not required.

Formal verification of MATLAB/Simulink and Stateflow models of moderate size can be accomplished by an automatic transformation to hybrid automata via the Hybrid System Interchange Format (HSIF) (cf. [1]).

CHARON A hierarchic automata model for the specification of behavior is provided by the modeling language CHARON [4]. Additionally, it provides a hierarchical architectural model, based on ROOM actor diagrams. CHARON's transitions are non-urgent and thus do not have to be fired instantaneous [6] which leads to delays when firing transitions. Sensing, computation, and actuation are assumed to be performed within one period. This disengagement from the zero execution time assumption enables a more realistic implementation. A so called *dynamic dependency graph* ensures that algebraic constraints which are evaluated dependent on the system's current discrete state are evaluated in the dependency order, i.e. an assignment is not processed before its right-hand variables are updated. [5] defines *refinement* for hybrid CHARON models.

Hybrid Bond Graphs In *bond graphs*, the physical behavior is modeled using different bond graph elements and active bonds between them to describe the exchange of power. The bond graph elements are simple, non domain-specific structures like resistances, capacities etc. Thus, bond graphs are applicable in different domains. Further, block diagram elements are used to specify the dynamics of a system. Therefore, bond graphs combine the power exchange and the dynamics within one diagram. In *hybrid bond graphs* [26, 27], so called *controlled junctions* are introduced into the bond graph. A finite state machine (FSM) is associated with each controlled junction. Each state of the FSM is of type on or off, indicating if the controlled junction acts like a normal junction or as a 0 value source. Thus, hybrid bond graphs are similar to the modeling concept of hybrid automata. The controlled junctions, used to blind out single elements, are similar to discrete or conditionally blocks in block diagrams.

HybridUML and HL^3 HybridUML [8] defines a profile for UML 2.0 which allows the specification of architecture and hybrid behavior, as well. Its formal semantics is defined by a transformation to the HL^3 language [8]. HL^3 models are implemented by a mapping to a multi-CPU computer system.

HyROOM, HyCharts, and Hybrid Sequence Charts HyROOM [35, 7] distinguishes between a hierarchical architectural model and a hierarchical behavioural model, similar to CHARON. The discrete states of the behavioural model are associated with MATLAB/Simulink blocks to specify hybrid behavior.

As described in [35], a HyROOM model can be mapped to a HyChart [18, 34] model. This consists of an architectural model and a behavioral model as well, but the continuous parts are not specified by MATLAB/Simulink blocks, but by ordinary differential equations. By adding so called *relaxations* to the state machine, the behavior becomes implementable, but additional behavior is added as well. The semantics of the models is defined formally, which enables verification. Requirements of the hybrid system can be specified by Hybrid Sequence Charts [17].

Masaccio and Giotto Within the Fresco project, the high-level modeling language Masaccio [20] has been developed. It builds complex components by the parallel and serial composition of atomic discrete and atomic continuous components. It defines the interface of a component which contains (among others) dependency relations, describing which continuous output signals depend on which continuous input signals.

The high-level programming language Giotto [22, 21] which is also part of the Fresco project is also based on components. Giotto models may be derived from Masaccio models. Similar to automata models, a Giotto program consists of modes and mode switches. The behavior of a mode is described by C code. Each mode is associated with a period, specifying how often its behavior is executed, and a worst-case execution time (WCET) for the according C code. Switch-frequencies specify how often a switch is evaluated. The Giotto program is automatically implemented for a so called virtual Embedded Machine that realizes the timing specifications for the target platform.

In [22], it is described how MATLAB/Simulink has been extended for the specification of Giotto programs: Reconfiguration is specified within such Giotto Simulink models by so called Giotto case blocks, switching between different parts of the Simulink model. Each combination of these case-dependent parts of the Simulink model build one of the modes.

MechatronicUML Hierarchical architectural, hierarchical behavioral models, and the notion of deployment diagram are provided by *Mechatronic UML* [11]. Be-

Approach	References	URL
MATLAB/Simulik/Stateflow	[1]	http://www.mathworks.com
Hybrid Bond Graphs	[26, 27]	http://moncs.cs.mcgill.ca/people/mosterman
CHARON	[4, 6, 5]	http://www.cis.upenn.edu/mobies/charon/
Masaccio and Giotto	[22, 21, 20]	http://www.eecs.berkeley.edu/~fresco
HybridUML, HL^3	[8]	http://www.informatik.uni-bremen.de/agbs/research/hybriduml/
HyROOM/HyCharts/Hybrid Sequence Charts	[35, 7, 18, 17]	http://www4.in.tum.de/~stauner/
MechatronicUML	[16, 11, 10]	http://www.fujaba.de/projects/realtime/
Ptolemy II	[23]	http://ptolemy.eecs.berkeley.edu/
SysML	[29, 31]	http://www.sysml.org
Time Weaver	[13]	http://www.escherinstitute.org/Tools/TimeWeaver.asp
UML^h	[15, 14]	http://swt.cs.tu-berlin.de/~nordwig/HYFOS/

Table 1. List of considered techniques and tools and the employed references

havior is specified by hierarchical hybrid refconfiguration charts which are extensions of UML statecharts and apply concepts of hybrid statecharts. Transitions in hybrid reconfiguration charts are associated with WCETs and deadlines. The WCETs respect that firing transitions, raising events, and executing side-effects consumes time and the deadlines take into account that recognizing activated transitions always occurs with a delay. This semantics disengages from the zero-execution time semantics, usually applied in automata models and thus is implementable.

Hybrid behavior is specified by associating configurations of hybrid components to the discrete states. This enables specification and modular verification of reconfiguration across multiple components [16]. Continuous behavior is specified by block diagrams.

Ptolemy II Similar to Charon, HybridUML, or HyROOM, Ptolemy II [23] distinguishes between architectural and behavioral design. In contrast to the aforementioned approaches, Ptolemy II provides different semantic domains (*models of computation*). Among others, it supports semantics for continuous time, discrete time, (distributed) discrete events, finite state machines, synchronous dataflow, and timed multitasking. Ptolemy II even supports combination and integration of these models of computation.

SysML The *Systems Modeling Language (SysML)* [31],[5] which is a distinguishing answer to OMG's request for proposals for *UML for System Engineering (UML for SE)* [29], and which is already adopted by the OMG, extends a subset of the UML 2.0 specification. One extension, related to the design of continuous and hybrid systems are so called *Assemblies* which are based on UML's structured classes. They describe the fine structure of a class extended by continuous communication links between ports. *Parametric Constraints* allow specifying parametric (arithmetic) relations between numerical attributes of instances. Continuous components could be modeled by defining the according differential equations by parametric constraints for a class. The nodes of activity diagrams are extended with continuous functions, in- and outputs, and flow information for the exchange of continuous data.

Time Weaver In [13], a framework for the component-based development of real-time systems is presented. The framework allows the specification of temporal annotations, like deadlines and periods, which enables schedulability analyses. Reconfiguration is specified by so called mode switches similar to discrete blocks in MATLAB/Simulink.

UML^h In UML^h [15], the architecture is specified by extended UML classes diagrams that distinguish between discrete, continuous and hybrid classes. Hybrid behavior is specified by a textual description of mathematic correlations between discrete and continuous variables, similar to the object-oriented extension of Z [14].

The findings with respect to the identified requirements are summarized in Table 2 and discussed and outlined in more detail in the following sections.

5. Modeling Support

All presented tools and techniques support the specification of architecture or structure by a notion of classes or component diagrams. *Mechatronic UML* and SysML are due to their UML 2.0 support the only ones that provide deployment descriptions.

All approaches allow the specification of continuous behavior by block diagrams, differential equations, or similar textual descriptions. Discrete behavior is specified by state machines, mostly with support for orthogonal states and history. Except SysML, they integrate continuous and discrete behavior, and thus enable the specification of hybrid reconfiguration. Nearly all approaches embed the continuous models in the discrete state machines – just in MATLAB/Simulink, hybrid bond graphs, and Time Weaver, these model elements are separated which leads just to a restricted *modular* reconfiguration. Specifying reconfigura-

5 http://www.sysml.org

Legend: X provided O not provided – not applicable Property\Approach	Techniques and Tools	MATLAB/Simulink/Stateflow	CHARON	Hybrid Bond Graphs	HybridUML/HL^3	UML^h	HyROOM/HyCharts/Hybrid Sequence Charts	Masaccio/Giotto	Mechatronic UML	Ptolemy II	SysML	Time Weaver
Used Basis Approaches												
Block Diagrams		X	O	O	O	O	O	O	X	X	O	O
Hybrid (I/O) Automata/Hybrid Statecharts		O	X	O	X	X	X	O	X	X	O	X
ROOM		O	X	O	O	O	X	O	O	X	O	O
UML 2.0		O	O	O	X	X	O	O	X	O	X	O
Modeling												
structure												
structure/composition		X	X	X	X	X	X	X	X	X	X	X
deployment		O	O	O	O	O	O	O	X	O	X	O
behavior:												
continuous		X	X	X	X	X	X	X	X	X	X	O
state machine		X	X	X	X	X	X	X	X	X	X	X
hybrid		X	X	X	X	X	X	X	X	X	O	X
hierarchical state machine		X	X	O	X	X	X	X	X	X	X	O
orthogonal states		X	X	O	X	X	O	X	X	X	X	O
history		X	O	O	X	X	X	O	X	O	X	O
sequence diagrams		O	O	O	O	O	X	O	X	O	X	O
real-time		-	-	-	-	-	X	-	X	-	X	-
hybrid		-	-	-	-	-	X	-	O	-	O	-
reconfiguration		X	X	X	X	X	X	X	X	X	O	X
embedding (E) or separated (S)		S	E	S	E	E	E	E	E	E	O	S
modular reconfiguration		O	X	O	X	X	X	X	X	X	O	O
support for intelligent resource management		O	O	O	O	O	O	O	X	O	O	O
modularity		X	X	X	X	X	X	X	X	X	X	X
interfaces/modules		X	X	X	X	X	X	X	X	X	X	X
interface sufficient to ensure correct embedding?		O	O	O	O	O	O	O	X	O	O	O
static (S) / dynamic (D) interfaces		S	S	S	S	S	S	S	D	S	S	S
MDD Support												
general:												
visual		X	X	X	X	X	X	X	X	X	X	X
standard (S) / standard extension (SE) / non standard (N) / extension of non standard (NE)		N	NE	O	NE	NE	NE	N	NE	NE	SE	
supported MDA levels												
PIM		X	X	X	X	X	X	X	X	X	X	X
PSM		O	O	O	O	O	O	X	O	O	O	O
Code		X	X	X	X	X	X	X	X	X	O	X
supported transformations/synthesis												
PIM -> PSM		O	O	O	O	O	O	X	O	O	O	O
PSM -> CODE		O	O	O	O	O	O	X	O	O	O	O
timing of the activation		-	-	-	-	-	-	X	-	-	O	O
correct timing including WCET		-	-	-	-	-	-	X	-	-	O	O
PIM -> CODE (direct mapping without PSM)		X	X	X	X	X	X	O	X	X	O	X
timing of the activation		O	X	O	O	O	X	-	X	O	O	O
correct timing including WCET		O	O	O	O	O	O	-	X	O	O	O
Support for formal analysis												
semantics		X	X	X	X	X	X	X	X	X	O	X
formal semantics		O	X	O	X	X	X	X	X	O	O	O
implementable real-time semantics		-	X	-	O	O	X	O	X	O	-	-
not idealized continuous behavior		-	O	-	O	O	X	O	O	O	-	-
refinement		O	X	O	O	O	X	O	X	O	O	O
simulation		X	X	X	O	O	X	O	X	X	O	O
schedulability analysis		O	X	O	O	O	O	X	X	X	O	X
automatic verification/model checking		O	X	O	O	O	O	X	X	O	O	O
real-time behavior		-	O	-	-	-	-	X	X	-	-	-
real-time behavior with reconfiguration		-	O	-	-	-	-	X	X	-	-	-
hybrid behavior		-	X	-	-	-	-	X	O	-	-	-
scalable (modular, compositional)		-	O	-	-	-	-	O	X	-	-	-

Table 2. Overview about the technologies and tools w.r.t the identified requirements

tion with SysML may be possible with workarounds, but concrete examples for this issue do not exist.

Mechatronic UML supports modeling of real-time scenarios with UML sequence diagrams and subsequent scenario-based synthesis of the role behavior. Hybrid Sequence Charts is the only approach that also supports hybrid scenarios.

Besides MATLAB/Simulink and Hybrid Bond Graphs, which do support only an instance view, all approaches support modular architecture and interface descriptions of the modules. *Mechatronic UML* is the only approach that supports dynamic interfaces which is required to ensure correct cross module reconfiguration. Further, *Mechatronic UML* supports intelligent resource management, i.e. it supports a flexible resource management system that dynamically shifts resources between applications [9].

Although most techniques and tools support modular design and modular reconfiguration, they do not respect that a module can change its interface due to reconfiguration which can lead to incorrect configurations.

6. MDD Support

All presented approaches are visual specification languages. CHARON, HybridUML, HL^3, UML^h, HyROOM, HyCharts, and Ptolemy II extend ROOM or UML and are based on hybrid automata or hybrid statecharts. Their models are located on the PIM level and enable code generation without a mapping to the intermediate PSM.

Hybrid Bond graphs are not a standard, but provide similar modeling facilities like block diagrams. It also provides models on the PIM level and enable a direct mapping to code. An intermediate PSM is not supported.

Masaccio models are located on the PIM level, Giotto models on the PSM level. As Giotto models might be derived from a Masaccio model, and as Giotto models can be mapped to code, these approaches support the MDA process. Giotto and Masaccio are proprietary.

Mechatronic UML models which are based on UML 2.0, hybrid statecharts, and block diagrams are also located on the PIM level and can be mapped directly to code. SysML which is an extension of a subset of the UML standard provides also models on the PIM level, but issues like code or PSM synthesis are not addressed. Time Weaver also provides a PIM and a code generator that operates on this PIM.

The evaluation shows that the MDA approach is not supported well by the techniques and tools. However, most of them have been developed without focus on MDA. Nevertheless, there is still need for the seamless support of transformations from PIM via PSM to code to benefit from the advantages of MDA. In real-time systems, it is additionally important to respect WCETs and the timing of the activation.

7. Support for Formal Analysis

CHARON, Masaccio, HybridUML, HL^3, UML^h, HyROOM, HyCharts and *Mechatronic UML* have a formally defined semantics, but due to the assumption of zero-execution times or zero-reaction times, most of them

are not implementable. Implementable semantics are provided by CHARON, Giotto, and *Mechatronic UML*. HyCharts are implementable after defining so called relaxations to the temporal specifications. They even respect that idealized continuous behavior is not implementable on discrete computer systems. Further, CHARON and *Mechatronic UML* provide a semantic definition of refinement. Ptolemy II even provides multiple semantics and supports their integration.

Automatic verification of the real-time behavior including the reconfiguration is supported by CHARON, and *Mechatronic UML*. CHARON even supports hybrid modelchecking. *Mechatronic UML* supports even compositional modelchecking of real-time properties which is also possible for CHARON and HyCharts in principle due to their definition of refinement. To the best of our knowledge, MATLAB/Simulink, CHARON, hybrid bond graphs, HyROOM, *Mechatronic UML*, and Ptolemy II are the only approaches that support simulation of the hybrid model. Schedulability analysis is supported by CHARON, Masaccio/Giotto, *Mechatronic UML*, Ptolemy II, and Time Weaver.

In order to guarantee a system's correctness, simulation and verification techniques are required. Although some tools and techniques provide such support, there is still need for modular or compositional approaches –especially for real-time or hybrid systems– in order to handle even systems of relevant size.

8. Summary & Future Work

We reviewed in this paper the capabilities of a number of existing techniques and tools for the visual model-driven development of networked, safety-critical embedded systems. A summary of these results are depicted in Table 2, where the more detailed results have been summarized using a somehow oversimplifying scheme. The presented survey is restricted to a number of tools which address a reasonable subset of the identified requirements. The specific characteristics and capabilities for the tools w.r.t. the model-driven development has been considered first. The offered modeling concepts and their expressiveness for the relevant structural and behavioral aspects of software intensive systems have been reviewed afterwards. Which foundations and techniques for the formal analysis of the systems are provided, has been subject of another section.

In the future, we want to extend this survey and therefore encourage any reader to send us proposals for additional requirements or not considered techniques and tools. We will respect the results of the survey when developing new approaches.

References

[1] A. Agrawal, G. Simon, and G. Karsai. Semantic Translation of Simulink/Stateflow models to Hybrid Automata using Graph Transformations. In *International Workshop on Graph Transformation and Visual Modeling Techniques, Barcelona, Spain*, 2004.

[2] P. Allen. The OMG's Model Driven Architecture. *Component Development Strategies, The Monthly Newsletter from the Cutter Information Corp. on Managing and Developing Component-Based Systems*, XII(1), January 2002.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(3-34), 1995.

[4] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical Hybrid Modeling of Embedded Systems. In *First Workshop on Embedded Software*, 2001.

[5] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional Refinement of Hierarchical Hybrid Systems. In *Proceedings of the Fourth International Conference on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 33–48. Springer Verlag, 2001.

[6] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 171–182. ACM Press, 2003.

[7] K. Bender, M. Broy, I. Peter, A. Pretschner, and T. Stauner. Model based development of hybrid systems. In *Modelling, Analysis, and Design of Hybrid Systems*, volume 279 of *Lecture Notes on Control and Information Sciences*, pages 37–52. Springer Verlag, July 2002.

[8] K. Berkenkötter, S. Bisanz, U. Hannemann, and J. Peleska. Executable HybridUML and its Application to Train Control Systems. In H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, editors, *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 145–173. Springer Verlag, 2004.

[9] S. Burmester, M. Gehrke, H. Giese, and S. Oberthür. Making Mechatronic Agents Resource-aware in order to Enable Safe Dynamic Resource Allocation. In B. Georgio, editor, *Proc. of Fourth ACM International Conference on Embedded Software 2004 (EMSOFT 2004), Pisa, Italy*, pages 175–183. ACM Press, September 2004.

[10] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Informatics in Control, Automation and Robotics*. Kluwer Academic Publishers, 2005. to appear.

[11] S. Burmester, H. Giese, and M. Tichy. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In U. Assmann, A. Rensink, and M. Aksit, editors, *Model Driven Architecture: Foundations and Ap-*

plications, volume 3599 of *Lecture Notes in Computer Science (LNCS)*, pages 47–61. Springer Verlag, August 2005.

[12] L. Carloni, M. D. D. Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. *Modeling Techniques, Programming Languages and Design Toolsets for Hybrid Systems*. Project IST-2001-38314 COLUMBUS - Design of Embedded Controllers for Safety Critical Systems, WPHS: Hybrid System Modeling, July 2004. Version: 0.2, Deliverable number: DHS4-5-6.

[13] D. de Niz and R. Rajkumar. Time weaver: a software-through-models framework for embedded real-time systems. In *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 133–143. ACM Press, 2003.

[14] V. Friesen, S. Jähnichen, and M. Weber. Specification of software controlling a discrete-continuous environment. In *Proceedings of the 1997 international conference on Software engineering, Boston, Massachusetts, United States*, 1997.

[15] V. Friesen, A. Nordwig, and M. Weber. Object-Oriented Specification of Hybrid Systems Using UMLh and ZimOO. In *Proceedings of the 11th International Conference of Z Users on The Z Formal Specification Notation, Berlin, Germany*, volume 1493 of *Lecture Notes in Computer Science (LNCS)*, pages 328–346. Springer Verlag, 1998.

[16] H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*, pages 179–188. ACM Press, November 2004.

[17] R. Grosu, I. Krüger, and T. Stauner. Hybrid Sequence Charts. In *Proc. of the 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2000)*. IEEE, 2000.

[18] R. Grosu, T. Stauner, and M. Broy. A Modular Visual Model for Hybrid Systems. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'98)*. Springer Verlag, 1998.

[19] D. Henriksson, O. Redell, J. El-Khoury, M. Törngren, and K.-E. Årzén. Tools for Real-Time Control Systems Co-Design — A Survey. Technical Report ISRN LUTFD2/TFRT--7612--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, April 2005.

[20] T. A. Henzinger. Masaccio: A formal model for embedded components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS), Lecture Notes in Computer Science 1872, Springer-Verlag, 2000, pp. 549-563.*, 2000.

[21] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A Time-triggered Language for Embedded Programming. In *Proceedings of the IEEE 91:84-99, 2003. A preliminary version appeared in the Proceedings of the First International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2211, Springer-Verlag, pp. 166-184.*, 2001.

[22] T. A. Henzinger, C. M. Kirsch, M. A. Sanvido, and W. Pree. From Control Models to Real-Time Code Using Giotto. In *IEEE Control Systems Magazine 23(1):50-64, 2003. A preliminary report on this work appeared in C.M. Kirsch, M.A.A. Sanvido, T.A. Henzinger, and W. Pree, A Giotto-based helicopter control system, Proceedings of the Second International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2491, Springer-Verlag, 2002, pp. 46-60.*, 2002.

[23] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the Ptolemy Project. TechReport UCB/ERL M03/25, Department of Electrical Engineering and Computer Science, University of California, Berkeley, July 2003.

[24] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of Lecture Notes in Computer Science, Springer Verlag, 1992.

[25] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O Automata Revisited. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Rome, Italy, March 28-30, 2001*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer Verlag, 2001.

[26] P. J. Mosterman and G. Biswas. Modeling Discontinuous Behavior with Hybrid Bond Graphs. In *Proc. of the Intl. Conference on Qualitative Reasoning, Amsterdam, the Netherlands*, pages 139–147, May 1995.

[27] P. J. Mosterman and G. Biswas. A Theory of Discontinuities in Physical System Models. *Journal of the Franklin Institute*, 334B(6):401–439, January 1998.

[28] D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach. Self-Adaptive Software for Hard Real-Time Environments. *IEEE Inteligent Systems*, 14(4), July/August 1999.

[29] Object Management Group. *UML for System Engineering Request for Proposal, ad/03-03-41*, March 2003.

[30] Object Management Group. *UML 2.0 Superstructure Specification*, October 2004. Document: ptc/04-10-02 (convenience document).

[31] Object Management Group. *Systems Modeling Language (SysML) Specification*, January 2005. Document ad/05-01-03.

[32] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, May/June 1999.

[33] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons, Inc., 1994.

[34] T. Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technische Universität München, 2001.

[35] T. Stauner, A. Pretschner, and I. Péter. Approaching a Discrete-Continuous UML: Tool Support and Formalization. In *Proc. UML'2001 workshop on Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists*, pages 242–257, Toronto, Canada, October 2001.

[36] J. Sztipanovits, G. Karsai, and T. Bapty. Self-adaptive software for signal processing. *Commun. ACM*, 41(5):66–73, 1998.