Available online at www.sciencedirect.com
ScienceDirect
Journal of Visual Languages and Computing
17 (2006) 528–550

Journal of
Visual Languages
& Computing

www.elsevier.com/locate/jvlc

A survey of approaches for the visual model-driven development of next generation software-intensive systems

Holger Giese*, Stefan Henkler

Software Engineering Group, University of Paderborn, Warburger Str. 100, D-33098 Paderborn, Germany

Abstract

Software-intensive systems of the future are expected to be highly distributed and to exhibit adaptive and anticipatory behavior when operating in highly dynamic environments and interfacing with the physical world. Therefore, visual modeling techniques to address these software-intensive systems require a mix of models from a multitude of disciplines such as software engineering, control engineering, and business process engineering. As in this concert of techniques software provides the most flexible element, the integration of these different views can be expected to happen in the software. The software thus includes complex information processing capabilities as well as hard real-time coordination between distributed technical systems and computers.

In this article, we identify a number of general requirements for the visual model-driven specification of next generation software-intensive systems. As business process engineering and software engineering are well integrated areas and in order to keep this survey focused, we restrict our attention here to approaches for the visual model-driven development of adaptable software-intensive systems where the integration of software engineering with control engineering concepts and safety issues are important. In this survey, we identify requirements and use them to classify and characterize a number of approaches that can be employed for the development of the considered class of software-intensive systems.

Keywords: Survey; Software-intensive systems; Adaptable; MDD; MDA; Visual modeling

*Corresponding author.
E-mail addresses: hg@uni-paderborn.de (H. Giese), shenkler@uni-paderborn.de (S. Henkler).

1045-926X/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2006.10.002

1. Introduction

Today, for a rapidly growing range of products and services from many economic sectors software has become a key factor (cf. [1]). These systems belong to the category of software-intensive systems, "where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole" [2]. Such software-intensive systems today cover so diverse areas such as banking, communications, transportation, or medicine and many activities of our daily lives depend to a great extent on the proper operation of these systems.

The observation that creating software-intensive systems is extremely difficult is as old as the software engineering discipline itself and has been already raised during the historic NATO Software Engineering Conference 1968 in Garmisch (cf. [3]). While traditionally the involved disciplines such as control engineering, software engineering, and business process engineering as well as the integrating systems engineering discipline are often only operated in a weakly connected manner, today a more tight integration is often practiced. Software engineers, for example, have to participate in classical systems engineering activities such as establishing system requirements or systematically exploring design alternatives as software has a great impact on these tasks.

The next generation: The development challenge for software-intensive system of the future becomes even more demanding, as the next generation of software-intensive systems will be highly distributed, will exhibit adaptive and anticipatory behavior by adjusting their structure dynamically, and will act in highly dynamic environments interfacing with the physical world (cf. [1]). To master the often large and complex systems, a systematic engineering approach which includes modeling as an essential design activity is required. Therefore, a model-driven development (MDD) approach which provides a mix of models from a multitude of disciplines such as software engineering, control engineering, and business process engineering is required for mastering the complexity of these software-intensive systems. There is thus an urgent need for a new software design paradigm to master this task.[1]

While the integration of business process engineering and software engineering is well understood, no widely accepted standard for the integration of software engineering with control engineering concepts exists. The required integration of these different views will probably happen within the software, which thus has to cover a wide spectrum of modeling requirements which include complex information processing capabilities as well as the hard real-time coordination of distributed technical systems and computers.

The different, often visual modeling techniques of the single disciplines, such as block diagrams in control and 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 full potential.

[1] [4]: "The highest ranked recommendation of the NSF workshop was to develop a new software design paradigm that recognizes that uncertainty and emergent, unanticipated behaviors are likely to be forever present in the software-intensive systems of the future due to their ultra-large, networked, distributed, and diffuse-control natures."

UML 2.0, for example, already includes concepts such as the components whose origin is the telecommunications domain (cf. ROOM [5] actors). Message Sequence Charts (MSC) and Specification and Description Language (SDL), which also finally found their way into the UML 2.0, are another example. The OMG also acknowledged the demand and therefore published a Request for Proposal (RFP) for *UML for Systems Engineering* (*UML for SE*) [6]. 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. Together with other approaches we will thus review the related proposal called *systems modeling language* (*SysML*) [7] (www.sysml.org) which takes a subset of the UML 2.0 [8] and extends it to also support continuous and hybrid systems.

Survey approach: In contrast to available surveys which focus on the control perspective and hybrid systems, such as [9], or the software/hardware mapping during co-design such as [10], in this paper we review approaches for their conceptual benefits and shortcomings. Thereby, we regard especially the support for visual MDD of software-intensive systems focusing on the expected requirements of future generations of software-intensive systems.

We are mainly interested in the potential and concepts rather than the effectiveness of the approach together with the provided tools as studied for example in [11] in the context of safety-critical systems. We thus evaluate the approaches only with respect to their support for software design beginning with the modeling and resulting in an implementation. We do not focus on modeling physical behavior or on specifying requirements for the software.

In this survey we are interested in qualitative differences which can hardly be evaluated in small-scale experiments as the relevant differences will often only show up when large-scale problems are considered. In contrast to studies which focus on the effectiveness of tools collecting quantitative data from small-scale experiments (cf. [11]), we therefore employ a large-scale problem (introduced later in Section 2) in order to exemplify why the chosen survey criteria are essential and to outline what limitations follow from the identified shortcomings of the different approaches.[2]

Outline: The survey is structured as follows: a running example, the basic foundations of the different considered domains, and the employed survey criteria are sketched in Section 2. Then, the considered approaches for the visual MDD of software-intensive systems are briefly characterized in Section 3. The available modeling concepts and their expressiveness including real-time and continuous behavior are then considered in Section 4. The specific aspect of support for the MDD is afterwards discussed in Section 5. In Section 6, the need and foundations and support for formal analysis for the approaches are reviewed. Finally, we summarize our findings.

2. Prerequisites

To exemplify the employed survey criteria, we at first introduce the running application example, then outline the foundations required for all relevant domains, and finally provide a first overview about the employed criteria.

[2]The results presented in this survey are therefore based on the information provided by the cited references or personal communication and we decided not to try to validate the results by any kind of experiments with the tools. Surveys with more experimental character which address some of the considered tools can be found, for example, in [9,10].

Application example: As a concrete example, the railcab research project[3] is considered which aims at using a passive track system with intelligent shuttles that operate individually and make independent and decentralized operational decisions. Shuttles either transport goods or up to approx. 10 passengers.

The example provides a whole collection of different requirements typical for advanced software-intensive systems. At the lowest level, sophisticated control algorithms have to be employed in order to control the physical behavior of the shuttles. Above this lowest level of control algorithms, a layer which provides the required higher-order control functionality and monitoring capabilities is required. While the lowest layer only exhibits continuous behavior, at this middle levels both continuous and discrete behavior is required. Finally, at the highest level of the system the hard real-time coordination of the autonomous shuttle subsystems requires only hard real-time discrete behavior.

In addition, the system is intended to self-optimize its behavior in different ways: shuttles build convoys at run-time in order to reduce the energy consumption and achieve higher average throughput and traveling speed, the modules for suspension and tilt of the shuttle exchange information with other shuttles in order to anticipate and compensate the disturbances on the tracks. Also within more low-level modules the potential of self-optimization in different forms is studied to improve the performance of the system. Therefore, this project is a good example for future software-intensive systems which will, as outlined in [1], "exhibit adaptive and anticipatory behavior; they will process knowledge and not only data, and change their structure dynamically." Again the system provides multiple challenging development tasks: at the middle level adaptation across a layered system architecture is required which may include reconfiguration of the structure and not only parameters. At the highest level, the structural reconfiguration of the system architecture has to be addressed and the integration of business processes for the logistics is required.

Involved domains and their specific techniques: As highlighted by the example, software-intensive systems are characterized by the need for system integration that involves control and system engineering, business process engineering, and software engineering.[4]

The standard notation for *system and control engineering* to structure and decompose their models are block diagrams, which are usually used to specify feedback-controllers as well as the controlled plant. In block diagrams, each block defines a relation between its input and output signals and directed arcs connect the input and output signals of the blocks. The dynamics of complex systems is classically described using differential equations or libraries of blocks with underlying continuous behavior described by differential equations. For each block the equations define the relation between its time-continuous input and output signals and state variables. In addition, time-continuous and value-discrete states and their modification might be specified using automata-like elements.

The standard approaches in *software engineering and business process engineering* world to describe data intensive aspects of the software or business are entity relationship diagrams and their object-oriented extension class diagrams as present in UML 1.X [12]. One of the first object-oriented notations with a technical background was ROOM [5]. Here, instead of class diagrams an architectural view of the system is defined in actor

[3] http://www-nbp.upb.de/en/index.html.

[4] For some introduction into the matter see [1,2,4,6,7].

diagrams that specify entities called actors, their communication interfaces, and the interconnections used for communication between the actors. Today, the defacto standard for software development is the UML. In its newest version, UML 2.0 [8], it supports both classes diagrams and component diagrams, which correspond to the ROOM concept of actor diagrams, to model the structure of a system.

Control engineering also includes discrete behavioral models, but pretty much focused on continuous models and therefore did not develop any concepts to cope with the increasing discrete complexity of today's software system. The *software engineering and computer science* community in contrast developed such models. However, the more expressive notations for state machines in software engineering such as ROOM charts as well as UML 2.0 state machines all extend Statecharts [13] and only support rather limited notions of time and no hybrid behavior. Hybrid automata [14] are one example of a formal model which bridges both worlds by assigning a continuous model (e.g. a feedback-controller) to each discrete state of an automaton. Related more expressive state machine models such as hybrid statecharts [15] in principle provide the required extensions of state machines by combining complex reactive behavior with continuous behavior. However, they have not found their way in any current approach.

Survey criteria: Taking the characteristics of the before outlined modeling notations of all domains relevant for software-intensive systems into account, we can identify in the following a number of general requirements for the visual MDD of software-intensive systems, which can be categorized into support for modeling, support for MDD, and support for the analysis of the models (cf. Sections 4, 5, and 6, resp.).

Support for *modeling* is a crucial prerequisite for the development of any complex systems. The developer needs support for appropriate abstraction and description techniques for the specific problems at hand. At first support for a structural view in essential to rule the complexity of the system. Furthermore, the common description techniques for the behavior which are state machines in the case of software engineering, some form of block diagrams for control engineering, and the activity view for business process engineering have to be supported. In addition, for requirement engineering often a scenario view is beneficial. As exemplified in the application example, advanced software-intensive systems further exhibit adaption. Finally, modularity is required to ensure a proper separation between different subsystems and facilitate maintainability.

As "models can serve as a vehicle for communicating information between business process engineers, control engineers and computer scientists" [1] and therefore can bridge the gap between the different disciplines, MDD is a promising approach to address the development of the next generation of software-intensive systems. To enable reuse or support multiple alternative platforms, we require a separation between platform-independent models (PIM), platform specific models (PSM) and the code level (in case of the software). The support for code generation can further be restricted to simulation purposes or provide production code.

Models "can also provide an additional basis for preimplementation validation of requirements and quality properties" [1] and thus *model analysis* can help to significantly improve quality and reduce development costs. A crucial prerequisite for analysis is a clear semantics as otherwise techniques such as simulation or formal verification have no solid ground to build upon. For complex systems the coverage due to simulation or testing can be rather limited, as in the case of the shuttle system where an overwhelming number of different system states make any reasonable coverage by means of such incomplete

Table 1
List of considered approaches and the employed references

Approach	References	URL	Abbreviation
CHARON	[16–18]	www.cis.upenn.edu/mobies/charon/	CHARON
HybridUML, HL3	[19]	www.informatik.uni-bremen.de/agbs/research/hybriduml/	HybridUML
HyROOM/HyCharts/Hybrid Sequence Charts	[20–23]	www4.in.tum.de/~stauner/	HyROOM
HyVisual and Ptolemy II	[24,25]	ptolemy.eecs.berkeley.edu/	HyVisual
Masaccio and Giotto	[26–28]	www.eecs.berkeley.edu/~fresco	Masaccio
MATLAB/Simulink/Stateflow	[29]	www.mathworks.com	MATLAB
Mechatronic UML	[30–34]	www.fujaba.de/projects/realtime/	MechUML
SysML	[6,7]	www.sysml.org	SysML
UML^h	[35,36]	swt.cs.tu-berlin.de/~nordwig/HYFOS/	UML^h

techniques impossible. In such cases, only scalable formal verification techniques can provide the required more complete coverage.

3. Considered visual model-driven approaches

Taking the identified criteria and the focus on the integration between software engineering and control engineering into account, we decided to consider approaches developed in the different domains, the state-of-the-art approaches employed today in practice, upcoming standards in the field, as well as those academic proposals which seem most advanced. Therefore, we selected the de facto standard in industry MATLAB/Simulink with Stateflow, the approaches CHARON, Masaccio and Giotto, HybridUML in combination with HL3, the triple HyROOM/HyCharts/Hybrid Sequence Charts, Mechatronic UML which have a computer science background, HyVisual/Ptolemy II which has a classical engineering background, and SysML[5] the upcoming standard of the OMG as potential candidates for the visual MDD for next generation software-intensive systems.[6] The name, the employed references, a related URL, and the employed abbreviation are summarized for each considered approach in Table 1.

CHARON: A hierarchic automata model for the specification of behavior is provided by the modeling language CHARON [16]. 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 [17] which leads to delays when firing transitions. Sensing, computation, and actuation are assumed to be performed within one

[5]As the standard is not finished yet and all the tools supporting the not finished standard do mix the approach with the UML 2.0 standard, we consider here only the standard and not toolsupport. This lead to a number of not supported requirements, as seen in the following sections.

[6]Please note that one author of the article has been a main contributor to Mechatronic UML.

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. Alur et al. [18] defines *refinement* for hybrid CHARON models.

CHARON's focus is the formal verification of hybrid systems based using discrete abstraction based on predicate abstraction.

HybridUML and HL3: HybridUML [19] defines a profile for UML 2.0 which allows the specification of architecture and hybrid behavior, as well. HybridUML inherits its formal semantics from CHARON and defines a transformation to the HL3 language [19]. HL3 models are implemented by a mapping to a multi-CPU computer system.

The idea of HybridUML is to provide for building safe systems. Thus, the operational area of HybridUML is the formal verification of hybrid systems.

HyROOM, *HyCharts*, *and Hybrid Sequence Charts*: HyROOM [20,21] distinguishes between a hierarchical architectural model and a hierarchical behavioral model, similar to CHARON. The discrete states of the behavioral model are associated with MATLAB/Simulink blocks to specify hybrid behavior.

As described in [20], a HyROOM model can be mapped to a HyChart [22,37] 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 with Hybrid Sequence Charts [23].

As CHARON and HybridUML the idea of HyROOM is to provide a formal semantics to enable verification.

HyVisual/Ptolemy II: Similar to CHARON, HybridUML, or HyROOM, Ptolemy II [24] distinguishes between architectural and behavioral design. In contrast to the aforementioned approaches, Ptolemy II support 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.

The Hybrid System Visual Modeler (HyVisual) is built on top of Ptolemy II. The domain of HyVisual is the simulation for continuous-time dynamical systems and hybrid systems.

The domain of both projects, HyVisual and Ptolemy II, is the modeling and simulation of real-time systems with focus on the assembly of concurrent systems.

Masaccio and Giotto: Within the Fresco project, the high-level modeling language Masaccio [28] 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 basis of Masaccio is the formal semantics for the domain of hybrid dynamical systems. The focus of Masaccio is to provide a formalized modeling language for verification.

The high-level programming language Giotto [26,27] 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 related 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.

Giotto is a framework for the time-triggered implementation of embedded systems. Its basic idea is the separation of platform-independent issues, like timing, from platform-dependent issues, like scheduling.

MATLAB/Simulink with Stateflow: The de facto industry standard employing block diagrams is MATLAB/Simulink and Stateflow. Here block diagrams and classical control engineering concepts have been integrated with a Statechart dialect named Stateflow. The system structures are all interconnected via time-continuous signals. The MATLAB/Simulink-based tool CheckMate can be used to verify MATLAB/Simulink and Stateflow models.

Besides the modeling aspects of MATLAB/Simulink the initial focus of this tools is simulation while other capabilities have been added later.

Mechatronic UML: Hierarchical architectural, hierarchical behavioral models, and the notion of deployment diagram are supported by Mechatronic UML [31]. The approach further provides an integration of continuous blocks which have been specified with the CAE tool CAMeL.[7] Discrete behavior is specified by hierarchical hybrid reconfiguration charts which are extensions of real-time statecharts and apply concepts of hybrid statecharts [38] and the continuous behavior is specified by block diagrams with the aforementioned CAE tool CAMeL. 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 is thus 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 [30].

SysML: The *SysML* [7] and [39],[8] which is an answer to OMG's REPs for *UML for SE* [6], 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 *Blocks* which are based on UML's composite 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.

[7] http://www.ixtronics.de/English/indexE.htm.

[8] http://www.sysml.org and http://syseng.omg.org/SysML.htm.

UML^h: UML^h [35] was developed in the context of the HYFOS project.[9] Two key points of the project are to develop a formal modeling approach for hybrid systems and the simulation of the specified models. UML^h was developed to satisfy the first key point. In UML^h, 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* [36].

Employed domain specific techniques: In Table 2, the relation between the considered approaches and the techniques mentioned in the preceding section are described.

While MATLAB/Simulink does only make use of block diagrams, HyVisual, HyROOM and Mechatronic UML support block diagrams as one alternative concept. Continuous behavior is described by some form of differential equation in all considered approaches.

UML 1.X has been chosen as platform by UML^h, ROOM has been extended by CHARON, HyROOM, and HyVisual, and the newer approaches such as HybridUML, Mechatronic UML, and SysML extend UML 2.0.

Besides MATLAB/Simulink and SysML, which only supports UML 2.0 State Machines, all approaches combine complex reactive behavior with continuous behavior using some notion of a hybrid automaton. The considered approaches either use ROOM charts (CHARON, HyROOM), or UML state machines (HybridUML, UML^h, Mechatronic UML) as starting point. MATLAB/Simulink supports hybrid models via the combination of Simulink and Stateflow.

4. Modeling support

Structure: For the modeling support of the considered approaches, we look in the structural view for means to describe the type, class diagram, component/agent/block, deployment, pattern, and task view as most of them are crucial prerequisites to model complex systems. All presented approaches support the specification of architecture or structure by a notion of classes, component or block diagrams. HybridUML and Mechatronic UML are due to their UML 2.0 support the only ones that provide deployment descriptions. Further Mechatronic UML is the only approach which support pattern.[10] It is remarkable that none of the approaches support a task view even though this view is standard for embedded real-time systems.

Continuous behavior: As models have to integrate concepts from classical software domains such as software engineering and business process engineering with control engineering, the approaches should support both continuous and discrete behavior. Consequently, all approaches allow the specification of continuous behavior by block diagrams, differential and/or algebraic equations, or similar textual descriptions.

Reactive behavior: To describe complex reactive behavior, a state machine-like notation with concepts such as hierarchical states, orthogonal states, and history are required. It results in time-discrete state-dependent behavior.

Complex state-dependent reactive behavior can be specified in all approaches by state machines, mostly with support for orthogonal states and hierarchy. While most approaches support the notion of clocks and hybrid behavior in the sense of hybrid

[9] http://swt.cs.tu-berlin.de/~nordwig/HYFOS/.

[10] Supported are so-called Coordination Pattern [38].

Table 2
Overview about the approaches w.r.t the identified basic modeling approach

Techniques	Approaches								
	MATLAB	CHARON	HybridUML	UML^h	HyROOM	Masaccio	MechUML	HyVisual	SysML
Block diagrams	X [29]	–	–	–	–	–	X [32]	X [25]	–
Differential equations	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X	X [25]	X [6]
UML	–	–	–	X [33]	–	–	–	–	–
ROOM	–	X [16]	–	–	X [20]	–	–	X [25]	–
UML 2.0	–	–	X [19]	–	–	–	X [32]	–	X [6]
Hybrid (I/O) automata/hybrid statecharts	–	X [16]	X [19]	X [33]	X [20]	X [28]	X [32]	X [25]	–

X, provided; –, not provided.

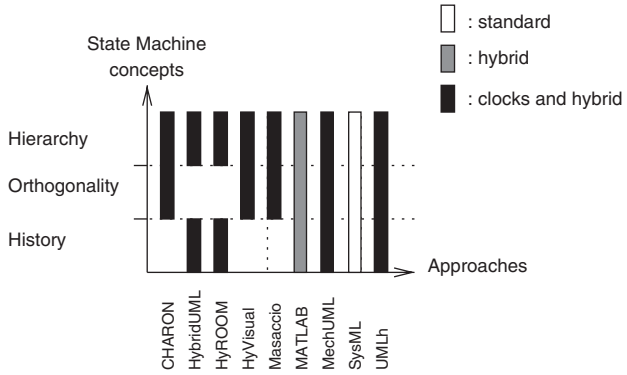

Fig. 1. Overview about the state machine concepts and the timing properties.

automaton, only MATLAB/Simulink and SysML do not explicitly offer these additional constructs. A summary of the supported concepts is depicted in Fig. 1.

Scenario view: Another often required description technique for behavior is the scenario view provided by notations such as timing diagrams or sequence diagrams. The description of appropriate coordination behavior at the highest level of the shuttle system would be an example where scenarios based descriptions would be helpful. Mechatronic UML and SysML both supports modeling of real-time scenarios with UML sequence diagrams and the standard time annotations. Hybrid Sequence Charts is the only approach that also supports hybrid scenarios.

Activity view: For the business process engineering an activity view as described by activity diagrams is also often required. In the shuttle system, the workflow modeling of the logistics would be one example. SysML and Mechatronic UML are the only approaches which support activity diagrams to model the behavior for control and object flows. While Mechatronic UML is restricted to standard activity diagrams and special extensions for structural reconfiguration, SysML supports also continuous data flows and therefore can describe hybrid behavior which is surprisingly not supported for state machines.

Adaptation: To build sophisticated software-intensive systems, the capability to adapt to the given context is of paramount importance. For the shuttle system, for example, adaptation or reconfiguration is in general required to adjust the control algorithms when shuttles build a convoy or when exploiting the context information to chose the most appropriate control strategy. If also self-optimization across multiple layers should be realized, reconfiguration cross module boundaries must be possible. Otherwise, the shuttle behavior as a whole cannot take advantage of the reconfiguration options of its submodules.

Nearly all approaches embed the continuous models in the discrete state machines using the hybrid automata concept in order to model reconfiguration of the continuous behavior within one module. Specifying reconfiguration with SysML should be possible with the activity diagrams, but concrete examples for this issue do not exist. In MATLAB/Simulink conditional blocks, which are only evaluated if explicitly triggered, can be used to model reconfiguration. Thus, a Stateflow diagram may be used to trigger the required elements of the currently active configuration. MATLAB/Simulink and Mechatronic UML alone

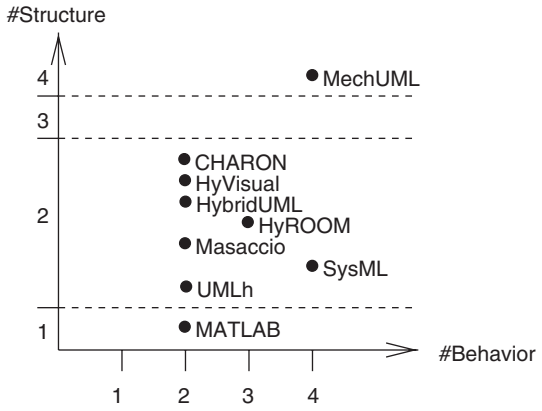

Fig. 2. Overview about the supported concepts for behavior and structure modeling.

permit that reconfiguration takes place across module/block boundaries. While in MATLAB/Simulink this can be achieved using conditional blocks and additional signals, in Mechatronic UML dynamic module interfaces are used to cover this issue.

Modularity: In complex systems, modularity is a major concern a suitable approach has to fulfill in order to scale and be maintainable. The support for modularity is considered looking whether an appropriate notion of interfaces and modules exist and whether these interface notions support reconfiguration across module boundaries. The development of the software of a complex software-intensive system like the shuttle system seems in fact not feasible, if the employed approach does not support any modularity concepts.

All approaches support modular architecture and interface descriptions of the modules. MATLAB/Simulink and Mechatronic UML are the only approaches that permit reconfiguration across module/block boundaries. However, only Mechatronic UML can ensure that complex reconfiguration steps across boundaries cannot result in an inconsistent reconfiguration step.

Summary: A general overview about the number of supported structural and behavioral notations is depicted in Fig. 2.[11] For the structural support we look therefore at the classes type, instance, deployment and pattern and for the behavior support we consider the classes continuous, state machine, scenario view and activity view. It is notable that MATLAB/Simulink does only support blocks and does not support block types as a concept. Most other approaches (CHARON, HyVisual, HybridUML, HyROOM, Masaccio, SysML, UML^h) also seem to focus on behavior and support only one structural notation, while only Mechatronic UML support all considered classes. Most approaches support continuous and reactive behavior and only HyROOM, Mechatronic UML, and SysML support additional notations such as sequence diagrams or activity diagrams.

An overview about all the different characteristics of the modeling support that have been taken into account in the discussion of the considered approaches can be found in Table 3.

[11]Please note that the numbers only do not provide any reasonable input for an assessment.

Table 3
Overview about the approaches w.r.t the identified modeling support

Support for modeling	Approaches								
	MATLAB	CHARON	HybridUML	UML^h	HyROOM	Masaccio	MechUML	HyVisual	SysML
Structure									
Instances and/or types	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Class diagram	–	–	X [19]	X [33]	–	–	X [31]	–	–
Component/agent/block	X [29]	X [16]	X [19]	–	X [20]	X [28]	X [31]	X [24]	X [6]
Deployment	–	–	–	–	–	–	X [31]	–	–
Pattern	–	–	–	–	–	–	X [31]	–	–
Process/task view	–	–	–	–	–	–	–	–	–
Behavior									
Continuous	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
State machine	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Hierarchical state machine	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Orthogonal states	X [29]	X [16]	–	X [33]	–	X [28]	X [31]	X [24]	X [6]
History	X [29]	–	X [19]	X [33]	X [20]	–	X [31]	–	X [6]
Timed	–	–	–	–	–	–	–	–	X [6]
Real-time	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	–
Hybrid	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	–
Scenario view	–	–	–	–	X [20]	–	X [33]	–	X [6]
Sequence diagrams	–	–	–	–	X [20]	–	X [33]	–	X [6]
Hierarchy	–	–	–	–	–	–	–	–	X [6]
Timed	–	–	–	–	–	–	–	–	X [6]
Real-time	–	–	–	–	X [20]	–	X [33]	–	–
Hybrid	–	–	–	–	X [20]	–	–	–	–
Activity view	–	–	–	–	–	–	X [34]	–	X [6]
Sequential	–	–	–	–	–	–	X [34]	–	X [6]
Concurrent	–	–	–	–	–	–	–	–	X [6]
Timed	–	–	–	–	–	–	–	–	–
Real-time	–	–	–	–	–	–	X [34]	–	X [6]
Hybrid	–	–	–	–	–	–	–	–	X [6]
Adaption	X [29]	X [16]	–	X [33]	X [20]	X [28]	X [31]	X [24]	–
Reconfiguration	X [29]	X [16]	–	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Reconfiguration across module/block boundaries	X [29]	–	–	–	–	–	X [31]	–	–
Modularity	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Interfaces/modules	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
Consistent reconfiguration across modules borders	–	–	–	–	–	–	X [31]	–	–

X, provided; –, not provided.

5. MDD support

MDD levels: The levels PIM, PSM, and code of the model-driven architecture (MDA) [40] correspond to important steps of MDD. Here, the question is which of these levels are populated in the approach. Depending on the supported levels, we can judge whether the approach can be employed in the earlier or later phases only or whether it supports the whole development phases.[12] In the shuttle system omitting a PIM would result in severe problems in the long run, as we cannot expect that all shuttles are put in operation on the same hardware and also the infrastructure will likely be subject to permanent updates of the hardware. Therefore, having a conceptual and platform-independent reference model for the system elements is in fact essential.

All approaches support the PIM level. HybridUML, Masaccio and Mechatronic UML supports additionally the PSM level. Surprisingly, SysML which is an extension of a subset of the UML standard provides only models at the PIM level as it focus on the requirement phase. As described in Section 3, HyROOM can be mapped to HyCharts models which are more detailed and formalized but still at the PIM level. HybridUML, Masaccio and Mechatronic UML support additionally the PSM level.

HybridUML models can be transformed to the PSMs of HL3. Giotto models which contain platform specific information can be derived from Masaccio models. Mechatronic UML supports the PSM level when WCET information is added to the statecharts.

Transformations/synthesis: An important set of criteria in the MDA view is also 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 consider "PIM → PSM", "PSM → CODE", and the direct synthesis "PIM → CODE".

As depicted in Fig. 3, all approaches despite SysML support the PIM level and provide either a direct PIM to Code step or a series of steps from PIM to PSM and then to Code (HybridUML, Masaccio and Mechatronic UML). CHARON, 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. In the HybridUML approach the HL3 models are implemented by a mapping to a multi-CPU computer system. Giotto models of the Masaccio approach can be mapped to code which is executable on a special virtual machine. Mechatronic UML PSM models, which are the PIM models with annotated WCET, can be mapped to code.

Code generation: For the code generation step several distinct cases have to be distinguished. Either the code is only meant to support simulation or it is possible to employ the code for real-time operation. In the former case more accurate numerical solvers might be employed which are at odds with the required hard real-time behavior. If the code is in principle applicable for real-time processing, it might further guarantee only the correct timing of the activation or in addition the correct schedulability.

As depicted in Fig. 3, UML^h and HyVisual/Ptolemy II support code generation for simulation purposes. Code generation of MATLAB/Simulink and HyROOM models are

[12]Using the MDA terminology for our survey criteria does not imply that the considered approaches must be conform with the MDA architecture. Proposals such as platform-based design [41,42] which do not consider the MDA terminology can be also described to some extent in this terminology.

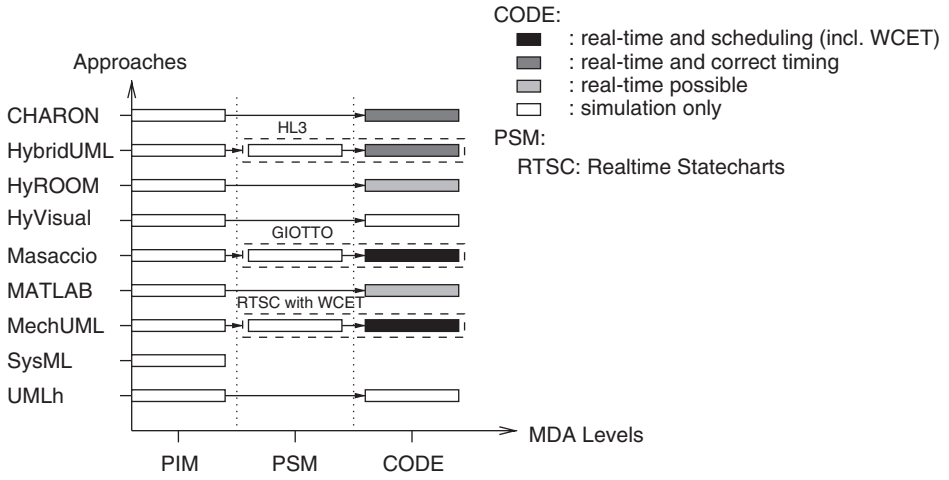

Fig. 3. Overview about the approaches w.r.t the identified MDD support.

possible with Realtime Workshop.[13] The resulting code may be used for real-time processing. CHARON and HybridUML support code generation with the correct timing of the activation. Masaccio and Mechatronic UML additionally support correct scheduling.

Summary: The overview presented in Table 4 shows that the MDD approach is not well supported by most of the approaches. Even though most of them have not been developed with focus on MDA, there is a need for the seamless support of transformations from PIM via PSM to code to be able to adjust the model to a specific platform details which might not be covered by direct PIM to code synthesis. Often additional legacy artifacts have to be taken into account in the PSM level, which is not supported even by the approaches which support some form of PSM model. Also only a small fraction of the approaches guarantees directly that the scheduling is correct by construction and that the timing of the activation is guaranteed. Instead, these properties often have to be checked for the generated code.

6. Support for model analysis

Semantics: As stressed in [9], a crucial prerequisite for a preimplementation validation of models is a full semantic integration of the employed visual modeling concepts and their underlying models to be able to make reasonable predictions. The current integration efforts at the OMG and by commercial vendors between the systems engineering and software engineering domains ignore this demand and only provide an informal semantics. However, as many software-intensive systems such as transportation, medical applications, and command and control systems are safety-critical systems, predictability is mandatory.

If a sound formal semantics exists, automated support for the analysis of the continuous and discrete behavior becomes possible. In addition, such a semantics is required to be implementable (and models should be implemented accordingly to this semantics) as otherwise the analysis result does not necessarily hold for the final system.

[13] http://www.mathworks.com/products/rtw/.

Table 4
Overview about the approaches w.r.t the identified MDD support

MDD support	Approaches								
	MATLAB	CHARON	HybridUML	UML^h	HyROOM	Masaccio	MechUML	HyVisual	SysML
MDD level									
PIM	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [31]	X [24]	X [6]
PSM	–	–	X [19]	–	–	X [26]	X [31]	–	–
Code	X [29]	X [17]	X [19]	X [33]	X [20]	X [26]	X [31]	X [24]	–
Transformations/synthesis									
PIM→PSM	–	–	X [19]	–	–	X [26]	X [31]	–	–
PSM→Code	–	–	X [19]	–	–	X [26]	X [31]	–	–
PIM→Code (directmapping without PSM)	X [29]	X [17]	–	X [33]	X [20]	–	–	X [24]	–
Code generation									
Real-time impossible (simulation only)	–	–	–	X [33]	–	–	–	X [24]	–
Real-time possible	X [29]	X [17]	X [19]	–	X [20]	X [26]	X [31]	–	–
Real-time + correct timing of activation	–	X [17]	X [19]	–	–	X [26]	X [31]	–	–
Real-time + correct scheduling of activation (incl. WCET)	–	–	–	–	–	X [26]	X [31]	–	–

X, provided; –, not provided.

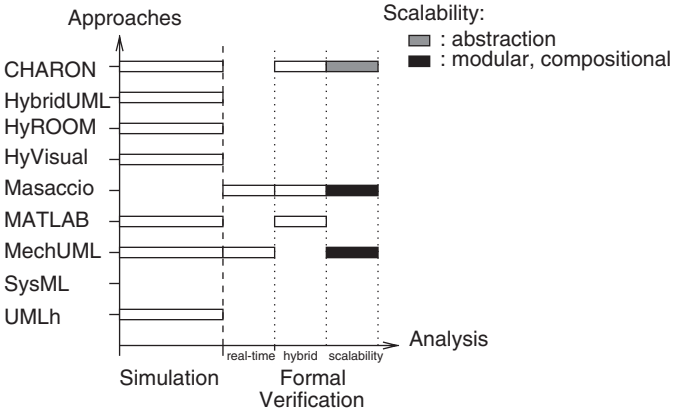

Fig. 4. Overview about the approaches w.r.t the identified support for analysis.

CHARON, Masaccio, HybridUML, HL3, UML^h, HyROOM, HyCharts and Mechatronic UML have 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 adding so-called relaxations. In contrast to the other approaches, HyCharts then even respect that idealized continuous behavior is not implementable on discrete computer systems. Furthermore, CHARON and Mechatronic UML provide a semantic definition of refinement. Ptolemy II even supports multiple semantics and supports their integration.

Simulation: 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. To the best of our knowledge, all approaches besides Masaccio support direct simulation of the hybrid model (cf. Fig. 4).

Schedulability analysis: A crucial prerequisite for hard real-time systems is a proof of the schedulability of the system. Such a schedulability analysis at the model level is supported by CHARON, Masaccio/Giotto, Mechatronic UML and Ptolemy II. For the other approaches the generated code has to be analyzed.

Formal verification: To further also ensure relevant functional and non-functional system properties beyond schedulability, we require techniques for the complete formal verification. 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 as otherwise no systems of practical relevance like the shuttle system can be tackled (cf. Fig. 4).

Automatic verification of the real-time behavior including the reconfiguration is supported by CHARON, Masaccio and Mechatronic UML. CHARON and Masaccio even supports model checking of hybrid models. Mechatronic UML and Masaccio support compositional and modular model checking of real-time properties employing refinement[14]

[14]The definition of *refinement* is informally as follows: a model *refines* another one, if it behaves always compliant with the behavior of the original model.

which is also possible for CHARON and HyCharts in principle as they also support a notion of refinement. CHARON additionally supports predicate abstraction as a means to improve the scalability of the verification.

Summary: The identified support for analysis can be found in Table 5. 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 improvement—especially for hybrid systems—in order to also handle practical problems of considerable complexity.

7. Conclusion

We reviewed in this article the capabilities of a number of existing model-driven approaches which can be employed for the visual MDD of software-intensive systems. As the "paradigm for software-intensive system development is still in its infancy" [1], we could not expect to find ready-made solutions which address all identified requirements. As the integration between business process engineering and software engineering is not considered a problem, we focused on the demands for adaptable, safety-critical embedded systems which include complex software. Thus, the integration of software engineering and control engineering was of uppermost importance.

The presented survey is restricted to a number of approaches which address a reasonable subset of the identified requirements. The specific characteristics and capabilities for the approaches w.r.t. the modeling has been considered first by analyzing the offered modeling concepts and their expressiveness for the relevant structural and behavioral aspects of software-intensive systems. Then, the support for the MDD has been reviewed and the support for formal analysis of the systems has been subject of the subsequent section.

The survey results are summarized using an oversimplifying scheme in Table 6. The table highlights the following observations: (1) Most approaches only support very restricted concepts for modeling such as the standard views for the structure and behavior. The scenario and activity views, which are important to address the early phases, are often not supported. (2) We can also conclude that all approaches lack full support for the PSM level which is required to integrate legacy elements or reuse elements, which is required for nearly all complex systems. (3) Another surprising observation is that most approaches, including Matlab/Simulink, do not support a code generation scheme which directly provides guarantees for any timing properties specified in the model. (4) Finally, a scalable approach for the formal verification of crucial safety properties is hardly supported.

We thus can conclude that most approaches are not able to fulfill the requirements identified for the support of the visual MDD of next generation software-intensive systems but rather address only particular aspects of the design challenge if at all. It remains an open question whether standardization efforts such as SysML will help to unite these often separated efforts. The efforts around the Hybrid Systems Interchange Format (HSIF)[15] may be also seen as a related approach, however the scope only address the hybrid behavior but no concept for complex structures. Experience with UML model exchange with XMI tells us that exchanging the models at the syntactic level alone is often not sufficient to integrate the different solutions. Another possible solution could be the support for multiple semantic domains as proposed by the Ptolomey II framework.

[15] http://www.isis.vanderbilt.edu/Projects/Mobies/downloads.asp#HSIF.

Table 5
Overview about the approaches w.r.t the identified support for analysis

Model analysis	Approaches								
	MATLAB	CHARON	HybridUML	UML[h]	HyROOM	Masaccio	MechUML	HyVisual	SysML
Semantics	X [29]	X [16]	X [19]	X [33]	X [20]	X [28]	X [30]	X [24]	–
Formal semantics	X[a]	X [16]	X [19]	X [33]	X [20]	X [28]	X [30]	X [24]	–
Implementable formal real-time semantics	–	X [17]	X [19]	–	X [20]	–	X [30]	–	–
Not idealized formal continuous semantics	–	–	X [19]	–	X [20]	–	–	–	–
Simulation	X [29]	X [16]	X [19]	X [33]	X [20]	–	X [30]	X [24]	–
Schedulability analysis	–	X [18]	–	–	–	X [27]	X [31]	X [24]	–
Formal verification/model checking	X[a]	X [17]	–	–	–	X [28]	X [30]	–	–
Model characteristics									
Real-time behavior	–	–	–	–	–	X [28]	X [30]	–	–
Real-time behavior with reconfiguration	–	–	–	–	–	X [28]	X [30]	–	–
Hybrid behavior	X[a]	X [17]	–	–	–	X [28]	–	–	–
Scalability									
Modular, compositional	–	–	–	–	–	X [28]	X [30]	–	–
Abstraction	–	X [18]	–	–	–	–	–	–	–

[a]Supported by CheckMate.

Table 6
Short overview about the approaches w.r.t the requirements

	Approaches								
	MATLAB	CHARON	HybridUML	UML^h	HyROOM	Masaccio	MechUML	HyVisual	SysML
Structure									
Instances and/or types	X	X	X	X	X	X	X	X	X
Deployment	–	–	–	–	–	–	X	–	–
Pattern	–	–	–	–	–	–	X	–	–
Process/task view	–	–	–	–	–	–	–	–	–
Behavior									
Continuous	X	X	X	X	X	X	X	X	X
State machine	X	X	X	X	X	X	X	X	X
Scenario view	–	–	–	–	X	–	X	–	X
Activity view	–	–	–	–	–	–	X	–	X
Adaption	X	X	–	X	X	X	X	X	–
Modularity	X	X	X	X	X	X	X	X	X
MDD level									
PIM	X	X	X	X	X	X	X	X	X
PSM	–	–	X	–	–	X	X	–	–
Code	X	X	X	X	X	X	X	X	–
Code generation									
Real-time impossible (simulation only)	–	–	–	X	–	–	–	X	–
Real-time possible	X	X	X	–	X	X	X	–	–
Real-time + correct timing of activation	–	X	X	–	–	X	X	–	–
Real-time + correct scheduling of activation (incl. WCET)	–	–	–	–	–	X	X	–	–
Semantics	X	X	X	X	X	X	X	X	–
Simulation	X	X	X	X	X	–	X	X	–
Schedulability analysis	–	X	–	–	–	X	X	X	–
Formal verification/model checking	X	X	–	–	–	X	X	–	–
Scalability	–	X	–	–	–	X	X	–	–

X, provided; –, not provided.

However, a solutions is required which is not only restricted to simulation but also provides comprehensible composition concepts and supports more advanced analysis techniques.

Acknowledgments

We thank Sven Burmester for his contributions to the related workshop paper, which served as a starting point for this one, and his comments on earlier versions of this article. Furthermore, we thank Florian Klein for his comments on earlier versions.

References

[1] M. Wirsing (Ed.), Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems, Edinburgh, GB, 2004.
[2] Standards Coordinating Committee of the IEEE Computer Society, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394, USA, Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE-Std-1471, 2000.
[3] P. Freeman, D. Hart, A science of design for software-intensive systems, Communications of the ACM 47 (8) (2004) 19–21.
[4] NSF Workshop on Modeling and Simulation for Design of Large Software-Intensive Systems: Toward a New Paradigm Recognizing Networked, Distributed and Diffuse-Control Software, 3–4 December, 2003.
[5] B. Selic, G. Gullekson, P. Ward, Real-Time Object-Oriented Modeling, Wiley, New York, 1994.
[6] Object Management Group, UML for System Engineering Request for Proposal, document ad/03-03-41, March 2003.
[7] SysML Partner, Systems Modeling Language (SysML) Specification, version 1.0 alpha, document ad/05-11-14, November 2005.
[8] Object Management Group, UML 2.0 Superstructure Specification, document: ptc/04-10-02 (convenience document), October 2004.
[9] L. Carloni, M.D.D. Benedetto, R. Passerone, A. Pinto, 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, version: 0.2, Deliverable number: DHS4-5-6, July 2004.
[10] D. Henriksson, O. Redell, J. El-Khoury, M. Törngren, 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.
[11] A.J. Kornecki, J. Zalewski, Experimental evaluation of software development tools for safety-critical real-time systems, Innovations in Systems and Software Engineering (2005) 176–188.
[12] Object Management Group, OMG Unified Modeling Language Specification, Version 1.5, OMG document formal/03-03-01, September 2001.
[13] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming 8 (3) (1987) 231–274.
[14] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theoretical Computer Science 138 (1995) 3–34.
[15] Y. Kesten, A. Pnueli, Timed and hybrid statecharts and their textual representation, in: J. Vytopil (Ed.), Formal Techniques in Real-Time and Fault-Tolerant Systems, Lecture Notes in Computer Science, vol. 571, Springer, Berlin, 1992.
[16] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, O. Sokolsky, Hierarchical hybrid modeling of embedded systems, in: First Workshop on Embedded Software, 2001.
[17] R. Alur, F. Ivancic, J. Kim, I. Lee, 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, ACM Press, New York, 2003, pp. 171–182.

[18] R. Alur, R. Grosu, I. Lee, O. Sokolsky, Compositional refinement of hierarchical hybrid systems, in: Proceedings of the Fourth International Conference on Hybrid Systems: Computation and Control (HSCC'01), Lecture Notes in Computer Science, vol. 2034, Springer, Berlin, 2001, pp. 33–48.
[19] K. Berkenkötter, S. Bisanz, U. Hannemann, 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, E. Westkämper (Eds.), Integration of Software Specification Techniques for Applications in Engineering, Lecture Notes in Computer Science, vol. 3147, Springer, Berlin, 2004, pp. 145–173.
[20] T. Stauner, A. Pretschner, I. Péter, Approaching a discrete-continuous UML: tool support and formalization, in: Proceedings of the UML'2001 Workshop on Practical UML-Based Rigorous Development Methods—Countering or Integrating the eXtremists, Toronto, Canada, 2001, pp. 242–257.
[21] K. Bender, M. Broy, I. Peter, A. Pretschner, T. Stauner, Model based development of hybrid systems, in: Modelling, Analysis, and Design of Hybrid Systems, Lecture Notes on Control and Information Sciences, vol. 279, Springer, Berlin, 2002, pp. 37–52.
[22] R. Grosu, T. Stauner, M. Broy, A modular visual model for hybrid systems, in: FTRTFT '98: Proceedings of the Fifth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, London, UK, 1998, pp. 75–91.
[23] R. Grosu, I. Krüger, T. Stauner, Hybrid sequence charts, in: Proceedings of the Third IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC 2000), IEEE, 2000.
[24] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, 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.
[25] C. Brooks, A. Cataldo, E.A. Lee, J. Liu, X. Liu, S. Neuendorffer, H. Zheng, HyVisual: a hybrid system visual modeler, University of California, Berkeley, Technical Memorandum UCB/ERL M05/24, 2005.
[26] T.A. Henzinger, C.M. Kirsch, M.A. Sanvido, W. Pree, From control models to real-time code using Giotto, IEEE Control Systems Magazine 23 (1) (2003) 50–64 (A preliminary report on this work appeared in: C.M. Kirsch, M.A.A. Sanvido, T.A. Henzinger, W. Pree, (Eds.), A Giotto-based Helicopter Control System, Proceedings of the Second International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science, vol. 2491, Springer, Berlin, 2002, pp. 46–60).
[27] T.A. Henzinger, B. Horowitz, C.M. Kirsch, Giotto: a time-triggered language for embedded programming, Proceedings of the IEEE 91 (2003) 84–99 (A preliminary version appeared in: Proceedings of the First International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science, vol. 2211, Springer, Berlin, 2001, pp. 166–184).
[28] 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, vol. 1872, Springer, Berlin, 2000, pp. 549–563.
[29] A. Agrawal, G. Simon, 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.
[30] H. Giese, S. Burmester, W. Schäfer, O. Oberschelp, Modular design and verification of component-based mechatronic systems with online-reconfiguration, in: Proceedings of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA, ACM Press, New York, 2004, pp. 179–188.
[31] S. Burmester, H. Giese, M. Tichy, Model-driven development of reconfigurable mechatronic systems with mechatronic UML, in: U. Assmann, A. Rensink, M. Aksit (Eds.), Model Driven Architecture: Foundations and Applications, Lecture Notes in Computer Science, vol. 3599, Springer, Berlin, 2005, pp. 47–61.
[32] S. Burmester, H. Giese, O. Oberschelp, Hybrid UML components for the design of complex self-optimizing mechatronic systems, in: Informatics in Control, Automation and Robotics, Kluwer Academic Publishers, Dordrecht, 2005.
[33] S. Burmester, H. Giese, F. Klein, Synthesis of parameterized UML real-time patterns from multiple parameterized real-timed scenarios, in: F. Bordeleau, S. Leue, T. Systä (Eds.), Scenarios: Models, Algorithms and Tools, Lecture Notes in Computer Science, vol. 3371, Springer, Berlin, 2005, pp. 193–211.
[34] S. Burmester, H. Giese, A. Seibel, M. Tichy, Worst-case execution time optimization of story patterns for hard real-time systems, in: Proceedings of the Third International Fujaba Days 2005, Paderborn, Germany, 2005, pp. 71–78.
[35] V. Friesen, A. Nordwig, 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, Lecture Notes in Computer Science, vol. 1493, Springer, Berlin, 1998, pp. 328–346.
[36] V. Friesen, S. Jähnichen, 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.
[37] T. Stauner, Systematic development of hybrid systems, Ph.D. Thesis, Technische Universität München, 2001.
[38] S. Burmester, H. Giese, M. Hirsch, Syntax and semantics of hybrid components, Technical Report tr-ri-05-264, University of Paderborn, October 2005.
[39] SysML Submission Team (SST), Systems Modeling Language (SysML) Specification, version 1.0 alpha, document ad/06-02-13, February 2005.
[40] P. Allen (Ed.), The OMG's Model Driven Architecture, Component Development Strategies, vol. XII, The Monthly Newsletter from the Cutter Information Corp. on Managing and Developing Component-Based Systems, 2002.
[41] A.S. Vincentelli, Defining platform-based design, EEDesign of EETimes, 2002.
[42] B. Bouyssounouse, J. Sifakis (Eds.), Embedded Systems Design: The ARTIST Roadmap for Research and Development, Lecture Notes in Computer Science, vol. 3436, Springer, GmbH, 2005.