Electronic Communications of the EASST
Volume 21 (2009)

Proceedings of the
3rd International Workshop on
Multi-Paradigm Modeling
(MPM 2009)

Model-Based Engineering of Supervisory Controllers using CIF

Ramon R.H. Schiffelers, Rolf J.M. Theunissen, Dirk A. van Beek, and Jacobus E. Rooda

10 pages

Guest Editors: T. Levendovszky, L. Lengyel, G. Karsai, C. Hardebolle
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Model-Based Engineering of Supervisory Controllers using CIF

Ramon R.H. Schiffelers, Rolf J.M. Theunissen, Dirk A. van Beek, and Jacobus E. Rooda [*]

(r.r.h.schiffelers, r.j.m.theunissen, d.a.v.beek, j.e.rooda)@tue.nl
Department of Mechanical Engineering
Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands

Abstract: In the Model-Based Engineering (MBE) paradigm, models are the core elements in the design process of a system from its requirements to the actual implementation of the system. By means of Supervisory Control Theory (SCT), supervisory controllers (supervisors) can be synthesized instead of designing them manually. In this paper, a framework based on the Compositional Interchange Format for hybrid systems (CIF) has been developed that integrates the MBE and the SCT paradigms. To illustrate the framework, an industrial-size case study has been performed: *synthesis of a supervisory controller for the patient support system of an MRI scanner*. In this case study, we address 1) modelling of the components and the control requirements; 2) synthesis of the supervisor; 3) simulation of the synthesized supervisor and a hybrid model of the plant; and 4) real-time, simulation based control of the supervisor and the actual patient support system of the MRI scanner.

Keywords: Model-based engineering, Supervisory control synthesis, Automata, Interchange formats

1 Introduction

Complex manufacturing machines consist of physical components (hardware) and control systems. The physical components, typically sensors, actuators and main structure, provide the means of the machine. The interactions between the physical components result in the so-called uncontrolled behavior of the machine. The control systems interact with the sensors and actuators to employ the means of the machine, which results in the controlled behavior of the machine. The controlled behavior should be such that the machine fulfills its functions, i.e. meets its pre-defined requirements. The control systems can be divided into five functional subsystems, see [PFC89]: 1) *Regulative control* (also known as direct or feedback control) that assures that the actuators reach the desired position in the desired way. 2) *Error-handling control* (also known as fault detection and isolation or exception handling) that detects erroneous behavior, determines the cause, and acts to recover the machine control system. 3) *Supervisory control* (also known as logic control) that coordinates the control of the individual machine components. This includes planning, scheduling and dispatching functions. 4) The *data processing subsystem* that stores and manipulates gathered data. 5) The *user interface subsystem* that allows the user to interact with the machine control system. In this paper, we focus on the development process of supervisory controllers (supervisors).

The current practice of developing supervisory controllers is to code them manually, based on (possibly informal) control requirements. Creating and changing requirements, a design and/or an implementation can be time consuming and error-prone. An other possibility is to use the *Model-Based Engineering* (MBE) paradigm, see [Ogr00, Bra08], in order to design the supervisory controller. In this case, (possibly formal) executable models for the supervisory controller are developed (by hand). Using analysis

[*] This work was partially done as part of the Darwin project under the responsibility of the Embedded Systems Institute, partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program, as part of the ITEA project Twins 05004, and as part of the Collaborative Project MULTIFORM, contract number FP7-ICT-224249

techniques such as simulation and verification, the system controlled by the supervisor can be analyzed. However, the development of a model of the supervisory controller is a highly non-trivial task. An alternative approach is to synthesize the supervisory controller automatically using *Supervisory Control Theory*, see [Won07, CL07]. First, the uncontrolled behavior of the machine to control is modeled precisely (by hand). Secondly, the requirements on the function of the machine (with respect to supervisory control) are modeled in detail (by hand). This includes safety and functional requirements. Out of these formal requirements and the model of the uncontrolled system, the supervisory controller can be synthesized automatically. This supervisory controller is proven correct by construction. This means that the controlled system behaves according to the prescribed requirements on the function of the machine and that the system is deadlock and lifelock free.

Although Supervisory Control Theory ensures that the controller is proven correct by construction, it remains a non-trivial task (but easier than the development of the supervisory controller itself) to define the correct plant and requirement models, and errors or undesired behavior might still exist in the plant models and/or requirement models.

Therefore, in this paper, we describe a framework developed for supervisory controller design. It combines the model-based engineering paradigm, that enables analysis by means of simulation and verification, together with supervisory control theory, that provides automatic synthesis of supervisors. For example, the plant models that are developed for the supervisor synthesis, can be reused for simulation of these plant models controlled by the synthesized supervisor. Simulation results can be used to validate whether the controlled plant behaves as intended. At a later stage in the design process, more detailed, e.g. timed or hybrid, plant models can be developed. Simulation of the more detailed plant models controlled by the synthesized model of the supervisor, enables a more detailed analysis. Early integration and testing can be performed by means of coupling models and realizations of different components via an infrastructure. Finally, the realization of all plant and controller components are integrated into one system.

To support the design process of industrial-size supervisory controllers, a (software) tool framework, based on the Common Interchange Formalism for hybrid systems, see [BRSR07, BRRS07], has been developed. To illustrate this tool framework, we describe an industrial-size case study that has been performed: *synthesis of a supervisory controller for the patient support system of an MRI scanner*.

The outline of this paper is as follows. Section 2 introduces the Model-Based Engineering (MBE) paradigm and Supervisory Control Theory (SCT). The developed framework and the accompanying tools are described in Section 3. The industrial-size case study is described in Section 4. Section 5 concludes the paper with conclusions and recommendations of future work.

2 MBE and SCT

This section provides the required background on Model-Based Engineering and Supervisory Control Theory.

2.1 Model-Based Engineering

The Model-Based Engineering (MBE) system development process is depicted in Figure 1[1]. It starts with defining system requirements R and creating a system design D. Based on system design the system is divided into n components. For each component $i \in \{1,\ldots,n\}$ requirements R_i are defined and its design D_i is developed. Based on the design of a component, models M_i are developed, each with their own purpose and required level of detail. After that, the component implemented resulting in a realization Z_i.

The models of the components can be used to perform further functional and performance analysis

[1] In the figure, the following conventions are used: icon 🗋 denotes documents, ○ denotes models, and □ denotes realizations.

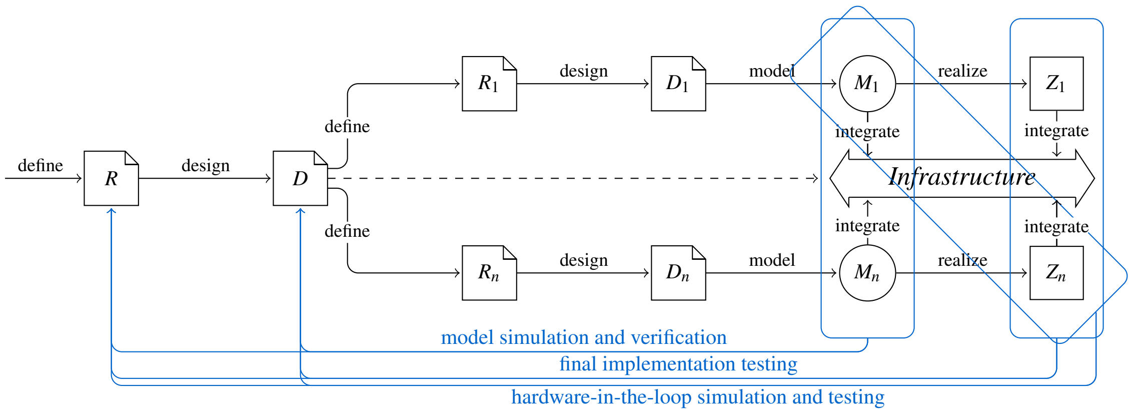


Figure 1: Model-Based Engineering.

by means of simulation and/or verification techniques such as model checking. Early integration and testing can be performed by means of coupling models and realizations of different components via an infrastructure. At the end, the realization of all components are integrated into one system. To confirm that the resulting system fulfills its requirements, the implemented components and the system as a whole are tested.

2.2 Supervisory Control Theory

Supervisory Control Theory (SCT) has been developed by W.M. Wonham and P.J. Ramadge, and their co-workers, in the 1980's, see [Won07, CL07]. It allows to synthesize the models of the supervisors, such that the correctness of these models is predetermined. The behavior of the system under control (further on, uncontrolled system) is considered unsatisfactory and has to be restricted by the supervisor to fulfill certain requirements. First, an uncontrolled system and its requirements are formally specified in terms of automata. Then, from these models, the supervisor is derived. The method guarantees that the system consisting of the derived supervisor and the uncontrolled subsystem fulfills the requirements. The theory has been implemented in a software package called TCT, see [Won07]. For large systems, the method suffers a state space explosion problem. To overcome this problem, research has been and is conducted to reduce complexity by using methods such as modular, decentralized and hierarchical control, see [CL07] and references therein.

3 Integrating MBE and SCT

In this section, we present the framework developed for supervisory controller design. It combines the model-based engineering paradigm, that enables analysis by means of simulation and verification, together with supervisory control theory, that provides automatic synthesis of supervisors. Furthermore, we present the developed tool framework that supports the design process.

3.1 Integrating MBE and SCT

Figure 2 shows the framework developed for supervisory controller design. From requirements $R_{S/P}$[2] of the controlled system, a design $D_{S/P}$ of the system is made and decomposed into a plant and a supervisory controller. Requirements R_S of the supervisor are formally modeled resulting in model M_{R_S} of the control requirements . From plant requirements R_P, a design D_P and one or more models M_P can be made, each

[2] Notation S/P denotes plant P under supervision of supervisor S.

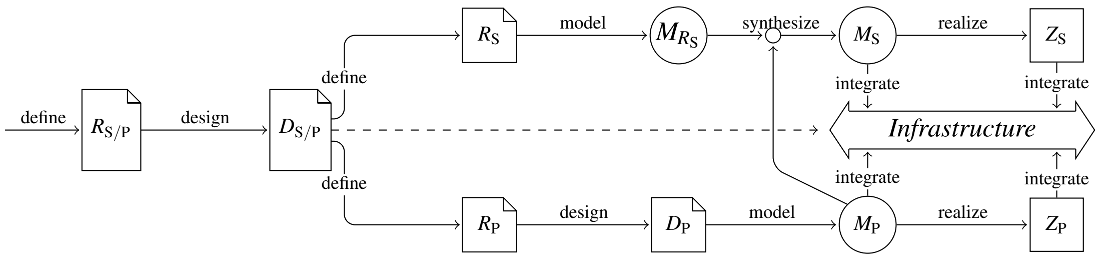


Figure 2: Supervisory controller design framework.

with a different level of detail. For instance, a discrete-event model of the plant can be made that serves as input for the supervisory controller synthesis, while a more detailed (possibly hybrid) model can be developed to study the dynamic behavior of the plant by means of simulation. In this way, simulation of the hybrid plant model and the model of the supervisor can reveal invalid assumptions in the models that are used for the supervisor synthesis. Using SCT, that takes as input the discrete-event model of the plant and the model of the control requirements for the supervisor, the model of supervisor M_S is synthesized. Using this model and the model(s) of the plant, the analysis techniques provided by the MBE paradigm can be used. By means of, for instance, code-generation, a realization of the supervisor can be made.

3.2 Tool framework

In this section we describe the tool-framework that is developed to support the development of supervisors using the integrated MBE and SCT paradigms. The tool-framework uses the Common Interchange Format for hybrid systems (CIF) to connect the controller synthesis tools and the analysis tools such as simulators, and modelcheckers.

3.2.1 The CIF language

The *Compositional Interchange Format* (CIF) for general hybrid systems, see [BRSR07, BRRS07, BRRS08], was recently developed within the European Network of Excellence HYCON, see [HYC05]. Its operational semantics, defined formally in a SOS style [Plo04], defines the *mathematical meaning* of a hybrid model and is independent of implementation issues and limitations, such as e.g. circular dependencies and algebraic loops. In [BRSR07], the CIF has been related to previous work on interchange formats for hybrid systems, such as [MoB02], and [PCPS06].

The CIF has been developed with two major purposes in mind 1) to provide a generic modeling formalism (and appropriate tools) for a wide range of general hybrid systems, and 2) to establish inter-operability of a wide range of tools by means of model transformations. The CIF serves as the basis of the European research project MULTIFORM, see [MUL08]. The main objective of this project is to develop interoperability of tools and methods based on different modeling formalisms to provide integrated coherent tool support for the design of large complex controlled systems. Within MULTIFORM, algorithms and tools for the translation to/from the CIF will be defined for a large variety of modeling languages, including CHI, GPROMS, MATLAB/SIMULINK, MODELICA, MUSCOD-II, PHAVER, and UPPAAL. In [SSB+09], the concepts of the CIF are illustrated by means of a hybrid model of a supermarket refrigeration system that exhibits both, nonlinear DAE dynamics as well as significant discrete dynamics, and serves as a challenging case study for hybrid control techniques in several European research projects. More information about CIF and CIF tools allowing, e.g., simulation and visualization, can be found in [Sys08].

3.2.2 Tool framework

Figure 3 shows the developed tool framework to support the supervisory controller design. Documents, models and realizations are graphically depicted according to the convention of Figure 1. (Software) tools are represented as filled, rounded rectangles.

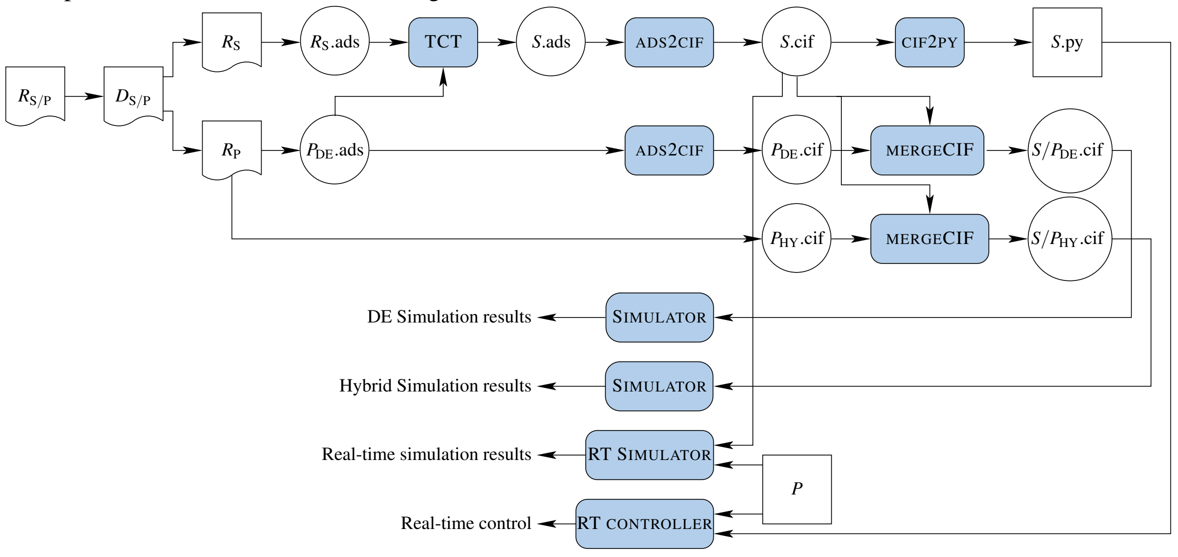


Figure 3: Tool framework based on CIF.

In the tool framework, $R_{S/P}$ and $D_{S/P}$ represent the requirements and the design of plant P under supervision of supervisor S, and R_P and R_S denote the requirement documents of the plant and supervisor, respectively. The models used in this figures are related to the models from Figure 2 as follows: $R_S.\text{ads} \in M_{R_S}$; $S.\text{ads}, S.\text{cif} \in M_S$; $P_{DE}.\text{ads}, P_{DE}.\text{cif}, P_{HY}.\text{cif} \in M_P$; and $S.\text{py} \in Z_S$[3].

The control requirements R_S are formally modeled by means of automata, resulting in R_S.ads. Model P_{DE}.ads describes the uncontrolled discrete-event behavior of the plant. The TCT tool takes as input (discrete-event) model of the uncontrolled system P_{DE}.ads and model of the control requirements R_S.ads. As output, we obtain model of the supervisor S.ads. Using the ADS2CIF translator, the models of the plant and the supervisor can be translated to equivalent CIF models (P_{DE}.cif, and S.cif, respectively). A discrete-event model of the plant controlled by the supervisor S/P_{DE}.cif is obtained by combining these individual component models using the MERGECIF tool.

At this moment, the translations are implemented using the general-purpose programming language Python. Recently, we have defined the conceptual model (domain model or meta model) of the CIF language by means of Ecore class diagrams [SBPM09]. These class diagrams (see [Sys08]) will be used for the model-to-model transformations that will be developed in near future.

Using the CIF simulator (SIMULATOR), the S/P_{DE}.cif model can be simulated to analyse its behavior with respect to the control requirements. After that, the discrete-event model of the plant can be replaced by the hybrid CIF model of the plant P_{HY}.cif. The next step is to replace the hybrid CIF model of the plant by actual hardware of the plant P. The real-time simulator (RT SIMULATOR) connects the hardware of the plant and the CIF model of the supervisor to analyse the response of the plant hardware as well as the simulation output. After that, using the CIF2PY compiler, the CIF model of the supervisor can be compiled into Python code S.py that can be executed on real-time control platform RT CONTROL that is connected to the actual hardware of the plant.

[3] The (file) extension '.ads' refers to the input language for the controller synthesis software package TCT, extention '.cif' refers to the CIF language, and extention '.py' refers to the Python language.

4 Case study: Patient Support System

An MRI scanner, see Figure 4, is used in medial diagnosis to render pictures of the inside of a patient non-invasively. To position a patient in an MRI scanner, a patient support system, consisting of a patient table (see Figure 5), a user interface and a light-visor is used.

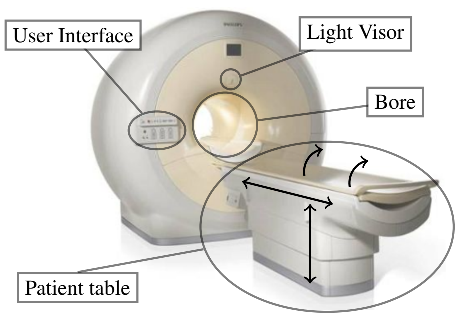


Figure 4: MRI scanner

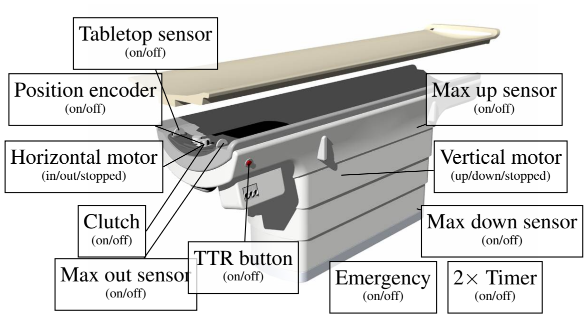


Figure 5: Patient table

The patient support system can be divided into the following components: vertical axis, horizontal axis and user interface. The vertical axis consists of a lift with appropriate motor drive and end-sensors. The horizontal axis contains a removable tabletop which can be moved in and out of the bore, either by hand or by means of a motor drive depending on the state of the clutch. It contains sensors to detect the presence of the tabletop, and the position of the tabletop. Furthermore, the system is equipped with hardware a safety system (emergency stop and tabletop release), that allows the operator to override the control system in emergency situations. Finally, the system contains a light-visor for marking the scan plane, and automated positioning of this scan plane to the center of the bore of the MRI scanner.

4.1 Controller synthesis

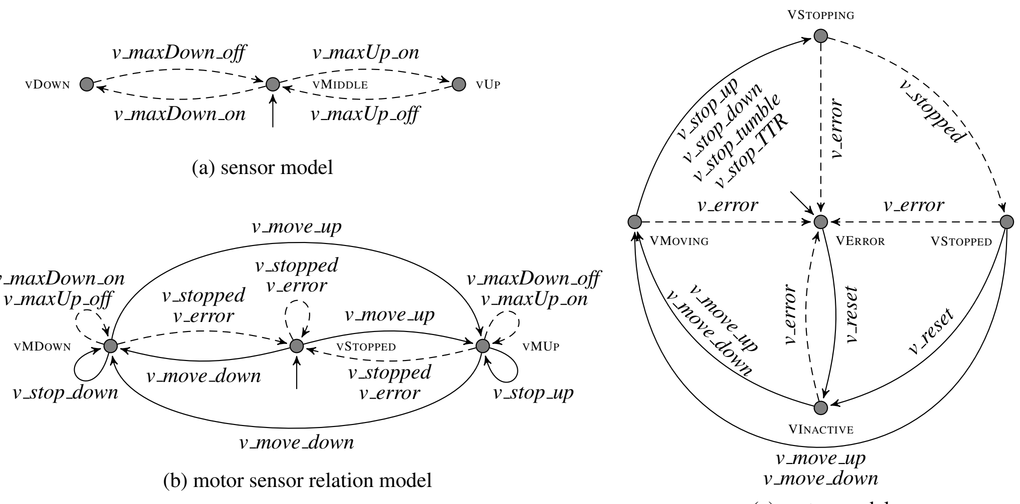
4.1.1 Models of the uncontrolled plant

The model of the uncontrolled plant consists of 5 components: 1) *vertical axis* consisting of the motor model, the sensors model, and the model of the relation between the motor and the sensors; 2) *horizontal axis* consisting of separate models for the motor, clutch, sensor, TTR sensor, TT sensor, encoder, and the TTS mode, respectively, and a model for the relation between the actuators and the sensors; 3) *PICU* consisting of models for the tumble switch, manual button, light-visor button, TTS button, manual LED, TTS LED, and the emergency LED, respectively. Furthermore, it contains a model to describe the time-behavior of the tumble timer; 4) *light visor*; 5) *emergency subsystem*. The complete models of the vertical axis are shown in Figure 6[4].

The vertical axis contains two sensors: maximally up and maximally down. Initially the table is assumed to be neither up or down, so that both end sensors are inactive. The sensors emit the events *v_max... _on* or *v_max... _off*, when a sensor becomes active or ceases to be active, respectively (Fig. 6a). Because of the physical location, the sensors are never active at the same time.

The motor is initially in an error state (Fig. 6c). The motor returns to this state after each error. From the error state, the system can be reset, to enter the inactive state. If the system is inactive, movement can be started. When movement is stopped, the system enters the stopping state. If the system is not moving anymore, the motor emits the event *stopped*, and the motor enters the stopped state. From this state, either movement can be started, or the system can be reset to enter the inactive state.

[4] In the figure, solid and dashed edges denote controllable and uncontrollable events, respectively; all states are marked.



(a) sensor model

(b) motor sensor relation model

(c) motor model

Figure 6: Plant model of the vertical axis ($\in P_{\mathrm{DE}}$.ads).

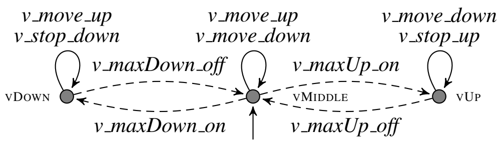


Figure 7: Example vertical control requirements ($\in S_{\mathrm{R}}$.ads).

The sensors do not change state when the motor is not moving (Fig. 6b). Only when the vertical motor is moving up, the maximally down sensor can turn *off* and the maximally up sensor can turn *on*, and likewise for the opposite direction. Note that this model is not a control requirement, it is an physical property of the system. If this model would be included as control requirement, the resulting supervisor would be empty due to restrictions on the uncontrollable sensor events, which cannot be realized by any supervisor. Summarized, the model of the uncontrolled plant consists of 27 relatively small, loosely coupled automata.

4.1.2 Models of the requirements

In total there are 57 automata describing the control requirements for the supervisor. The automaton model of the requirement "The vertical axis should not move beyond its maximally up and maximally down position" is shown in Figure 7. When the table is maximally up or down, it should stop (events *v_stop_up* and *v_stop_down*). In the maximally up position it is not allowed to move up (no event *v_move_up*), in the maximally down position it is not allowed to move down (no event *v_move_down*).

4.1.3 Synthesis of the supervisor

By using modular supervisor synthesis, see [Won07, CL07], 14 supervisors are synthesized which together implement all control requirements. The global nonblocking property is checked using automaton abstractions, see [SSRH08]. This abstraction procedure removes internal transitions of relevant automata, allowing the nonconflicting check to be performed over relatively small automata, even though the original system is fairly large.

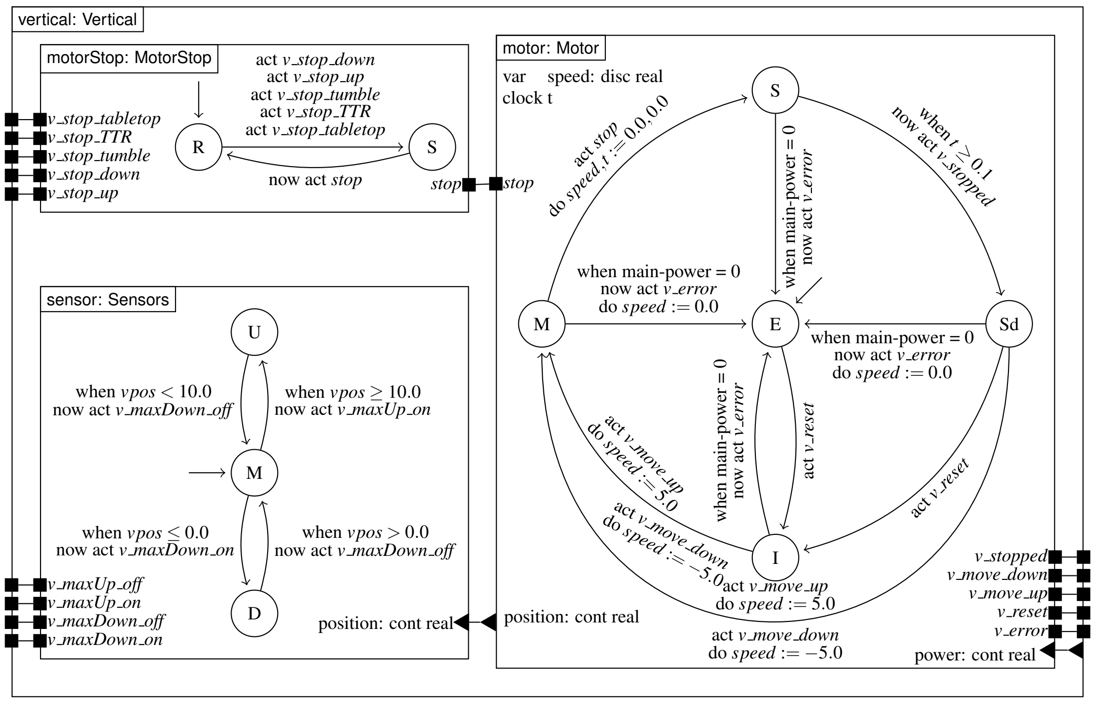


Figure 8: Hybrid CIF model of the vertical axis ($\in P_{\mathrm{HY}}$.cif).

4.2 Simulation of the supervisor and a hybrid model of the plant

Figure 8 shows the hybrid CIF model of the vertical axis component. In this figure, an automaton instantiation is represented as a solid box that is labeled with its name and the name of the automaton definition. Its internal declarations are listed in the upper left corner, and its external declarations are represented as ports on the borders of the box. The shape of a port depends on the type of declaration: a solid box denotes an action, and a solid triangle on the outer side (inner side) denotes an output (input) variable. Modes are visualized by means of circles labeled with the name of the mode. The flow, invariant and time-can-progress predicates are omitted from the figure, the predicates are true unless stated differently. Edges are represented as arrows between modes and are labeled with their guard, action, update (e.g. assignment to a variable), and, if the edge is urgent, the keyword *now*.

The CIF model of the vertical axis consists of an automata instantiation vertical that instantiates automaton definition Vertical. Automaton definition Vertical consists of the automata instantiations motorStop, motor, and sensor that instantiate the automata definitions MotorStop, Motor, and Sensor, respectively. Automaton definition MotorStop translates the various stop events from the supervisor to a single *stop* event; Motor models the dynamic behavior of the motor; and Sensors models the behavior of the sensors. The invariant predicate for all modes of the Motor automaton equals $\dot{position} = speed$, where $\dot{position}$ denotes the derivative of the *position*. The automata Motor and MotorStop synchronize on the *stop* event. The automata Motor and Sensors share the continuous variable *position*.

Figure 9 shows the simulation results of the following use case. Initially, the tabletop is positioned at the maximally out position, and the table is halfway up. The tumble switch is used to move the table to the upper position ($t = 2$). When the table reaches the upper position ($t = 3$), the table stops, and the tumble switch is released. Then the table is moved inward, first slowly ($t = 4$), then faster ($t = 5$). After that the movement is stopped ($t = 7$). Finally the table is moved out ($t = 8$), and switches automatically to moving down ($t = 10$) until it reaches the lowest position.

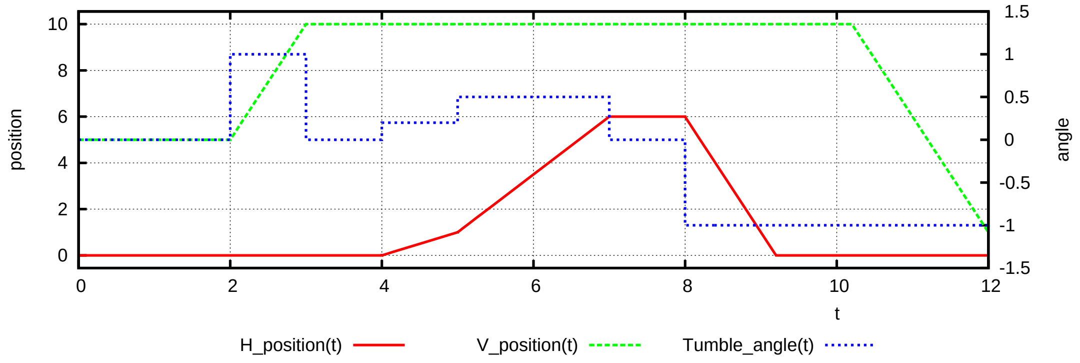


Figure 9: Simulation results S/P_{HY}.cif.

4.3 Real-time, simulation based control

The sensors and actuators of the actual patient support table are connected to an industrial grade I/O controller, which in turn is connected to a standard PC. The I/O controller conditions the sensor signals, translates sensor state changes to events, and translates events from the PC to appropriate inputs for the actuators. On the PC, the events from the I/O controller are buffered in an event queue. After receiving an event from the I/O controller, the state of the supervisor is updated, and the set of controllable events that is allowed by the supervisor is calculated. From this set, an event is selected and sent to the I/O controller. In the simulation model, the plant model and the model of the supervisor interact synchronously, i.e. they synchronize on common events. However, during the Real-time, simulation based control, the interaction between the patient table and the supervisor is asynchronously. More precisely: after a change of state of a sensor, this change has to be detected by the I/O controller (sensor polling delay). Then the I/O controller sends an event to the PC (communication delay between I/O controller and PC). After detection of the event (event queue polling delay), the state of the supervisor is updated, the allowed events are calculated, and an event is selected (calculation delay). In the setup, first all events from the event queue are processed. Then, when the event queue is empty, an allowed controllable event is selected and sent to the I/O controller.

5 Conclusions

In this paper, we developed a framework for supervisory controller design, based on the Model-Based Engineering and Supervisory Control Theory paradigms. This framework enables 1) simulation of the discrete-event plant models controlled by the synthesized supervisor model in order to validate the controlled behavior; 2) simulation of more detailed, e.g. timed or hybrid, plant models; 3) early integration and testing by means of real-time simulation; and 4) code-generation for the supervisory controller.

To support the design process of industrial-size supervisory controllers, a tool framework, based on the Common Interchange Formalism for hybrid systems (CIF) has been developed.

As part of the MULTIFORM project, the relations between the discrete-event model of the plant that is used for the supervisory controller synthesis and the more detailed timed and/or hybrid models of the plant will be studied.

Acknowledgements: The authors like to thank Dennis Hendriks and Albert Hofkamp for implementing major parts of the CIF tool framework, and Peter Thijs for implementing the ADS2CIF translation tool.

Bibliography

[Bra08] N. C. W. M. Braspenning. *Model-Based Integration and Testing of High-Tech Multi-disciplinary Systems*. PhD thesis, Eindhoven University of Technology, 2008.

[BRRS07] D. A. v. Beek, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers. Revised hybrid system interchange format. Technical report HYCON Deliverable D3.6.3, HYCON NoE, 2007.

[BRRS08] D. A. v. Beek, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers. Concrete syntax and semantics of the compositional interchange format for hybrid systems. In *17th Triennial World Congress of the International Federation of Automatic Control*. Pp. 7979–7986. Seoul, Korea, 2008.

[BRSR07] D. A. v. Beek, M. A. Reniers, R. R. H. Schiffelers, J. E. Rooda. Foundations of an Interchange Format for Hybrid Systems. In Bemporad et al. (eds.), *Hybrid Systems: Computation and Control, 10th International Workshop*. Lecture Notes in Computer Science 4416, pp. 587–600. Springer-Verlag, Pisa, 2007.

[CL07] C. G. Cassandras, S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2007.

[HYC05] HYCON Network of Excellence. http://www.ist-hycon.org/. 2005.

[MoB02] MoBIES team. HSIF Semantics. Technical report, University of Pennsylvania, 2002. internal document.

[MUL08] MULTIFORM consortium. Integrated Multi-formalism Tool Support for the Design of networked Embedded Control Systems MULTIFORM. http://www.multiform.bci.tu-dortmund.de, 2008.

[Ogr00] I. Ogren. On the principles for model-based systems engineering. *Systems Engineering* 3(1):38–49, 2000.

[PCPS06] A. Pinto, L. P. Carloni, R. Passerone, A. L. Sangiovanni-Vincentelli. Interchange Format for Hybrid Systems: Abstract Semantics. In Hespanha and Tiwari (eds.), *Hybrid Systems: Computation and Control, 9th International Workshop*. Lecture Notes in Computer Science 3927, pp. 491–506. Springer-Verlag, Santa Barbara, 2006.

[PFC89] R. J. Patton, P. M. Frank, R. N. Clarke (eds.). *Fault diagnosis in dynamic systems: theory and application*. Prentice-Hall, Inc., 1989.

[Plo04] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming* 60-61:17–139, 2004.

[SBPM09] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. *EMF Eclipse Modeling Framework*. Addison-Wesley, 2009.

[SSB$^+$09] C. Sonntag, R. R. H. Schiffelers, D. A. v. Beek, J. E. Rooda, S. Engell. Modeling and Simulation using the Compositional Interchange Format for Hybrid Systems. In Troch and Breitenecker (eds.), *6th International Conference on Mathematical Modelling*. Vienna, Austria, 2009.

[SSRH08] R. Su, J. H. van Schuppen, J. E. Rooda, A. T. Hofkamp. Nonconflict Check by Using Sequential Automaton Abstractions. Technical report 2008-010, Eindhoven University of Technology, 2008.
http://se.wtb.tue.nl/sereports

[Sys08] Systems Engineering Group TU/e. CIF toolset. http://se.wtb.tue.nl/sewiki/cif, 2008.

[Won07] W. Wonham. *Supervisory control of discrete-event systems*. Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2007.
http://www.control.toronto.edu/DES/