Integrating Model-Based Engineering and State-Based Supervisory Controller Synthesis

R.R.H. Schiffelers, D. Hendriks, R.J.M. Theunissen, D.A. van Beek, and J.E. Rooda *

Eindhoven University of Technology (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.
{R.R.H.Schiffelers,D.Hendriks,R.J.M.Theunissen,D.A.v.Beek,J.E.Rooda}@tue.nl

Abstract. Recently, a MBE-SCT framework has been developed that integrates the Model-Based Engineering (MBE) and the Supervisory Control Theory (SCT) paradigms. For event-based controller synthesis, an event-based tool framework, based on the Compositional Interchange Format for hybrid systems (CIF), has been developed that is an instantiation of the general MBE-SCT framework. In this paper, we developed a state-based tool framework to include a new SCT method called State-based Supervisory Synthesis, which is based on State-Tree Structures (STS). For this purpose, we developed formal translations 1) from (a subset of) STS to CIF; and 2) from the synthesized supervisor (that is encoded using Binary-Decision Diagrams) to CIF. The integration of the implementations of these translations and elements from the event-based tool framework forms the state-based tool framework. To illustrate the state-based tool framework, an industrial-size case study has been performed: *stated-based synthesis of a supervisory controller for the patient support system of an MRI scanner.*

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.

* This work was partially done as part of the European Community's Seventh Framework Programme (FP7/2007-2013) projects MULTIFORM and C4C, contract numbers FP7-ICT-224249 and FP7-ICT-223844, respectively, as part of the European Community's EUREKA cluster program ITEA 2 project Twins 05004, and 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.

meets its pre-defined requirements. The control systems can be divided into five functional subsystems, see [1]: 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 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 [2], [3], in order to design the supervisory controller. In this case, (formal, executable) models for the supervisory controller are developed (by hand). Using analysis techniques such as simulation and verification, the system controlled by the supervisor can be analyzed. A new, alternative approach is to synthesize the supervisory controller automatically using *Supervisory Control Theory*, see [4]. First, the uncontrolled behavior of the machine to control is modeled. Secondly, the (supervisory) control requirements are modeled. These requirements include safety and functional requirements. Out of these formal requirements and the model of the uncontrolled system, the supervisory controller can be automatically synthesized. 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. In [5], a MBE-SCT framework has been developed for supervisory controller design. It combines the model-based engineering paradigm, that enables analysis by means of simulation and verification, with supervisory control theory, that provides automatically synthesis of supervisors. To support the design process, a (event-based) tool framework, based on the Common Interchange Formalism for hybrid systems, see [6, 7] and event-based controller synthesis tools has been developed.

Recently, a new SCT method called State-based Supervisory Synthesis has been developed, see [8, 9]. The method is based on State-Tree Structures (STS). In this paper, we developed a state-based tool framework to integrate the State-based Supervisory Synthesis method in the MBE-SCT framework. It comprises a formal definition of the translation from (a subset of) STS to CIF and a formal definition of the translation from the synthesized supervisor (that is encoded using Binary-Decision Diagrams (BDDs)) to CIF. The integration of the implementation of these translations and (analysis) tools from the event-based tool framework forms the state-based tool framework. To illustrate the extension of the state-based tool framework, we describe an industrial-size case study that

has been performed: *stated-based 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 state-based supervisory controller synthesis. A brief overview of the MBE-SCT framework for supervisory controller design is given in Section 3. The state-based tool framework is described in Section 3.2, and the formal definitions of the developed tools are given in Section 4. The industrial-size case study is described in Section 5. Section 6 concludes the paper with concluding remarks.

2 State-based Supervisory Controller Synthesis

In this section, an overview of the state-based supervisory controller synthesis, based on STS is presented. Details and a formal description may be found [8, 9], whereas Section 4.1 gives (a simplified) definition of STS.

The system state space of an STS is organized as a state tree that consists of states and edges between them. There are three different kinds of states: 1) simple states that consist of a single state; 2) AND superstates consisting of child states executed in parallel; and 3) OR superstates consisting of child states, one of them being active at the time. The transition structure between the child states of an OR superstate is specified by means of a so-called holon (automaton). Examples of STS models are shown in Figure 7 and Figure 8.

The control requirements are state-based and specified using logical expressions. Two types of logical expressions can be used:

- *state exclusion*, requiring that a certain combination of states is never active at the same time;
- *state-transition exclusion*, requiring that a certain event does not occur when a certain combination of states is active.

The result of the supervisory controller synthesis is, for each controllable event, a control function which defines in which (combination of) states the event is enabled.

Example of a state-transition exclusion Using a state-transition exclusion, events can be disabled at certain states. In an event-based specification, events can be disabled by allowing the events in all states, except the one in which it must be disabled.

Consider for example, the requirement: "Movement may not be started if the sensor (see Figure 1) is on". This can be modeled by the state-transition exclusion $\{\textsc{SensorOn}\} \nrightarrow \mathit{move}$. To model this in an event-based specification, the specification is reversed: "Only when the sensor is off, movement may start". This can be modeled by taking the sensor model and adding a self-loop at the SensorOff state, see Figure 2.

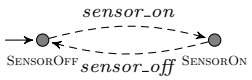

Fig. 1: Sensor model.

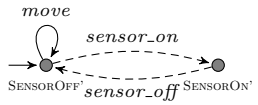

Fig. 2: Event-based state-transition exclusion.

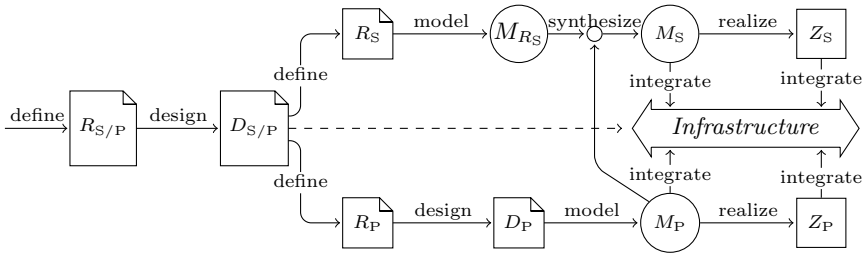

Fig. 3: Supervisory controller design framework.

3 MBE-SCT framework

In [5], a framework for supervisory controller design has been developed (MBE-SCT framework), see Figure 3[1]. The framework combines the model-based engineering paradigm, that enables analysis by means of simulation and verification, with supervisory control theory, that provides automatic synthesis of supervisors. 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 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 which 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.

In [5], the MBE-SCT framework has been instantiated with tools for event-based controller synthesis. The tool-framework uses the *Compositional Inter-*

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

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

change Format (CIF) for hybrid systems, see [6, 7, 10] connect the controller synthesis tools and the analysis tools such as simulators, model-checkers, etc. The CIF language, was recently developed within the European Network of Excellence HYCON, see [11]. Its formal semantics defines the *mathematical meaning* of a hybrid model and is independent of implementation issues and limitations. 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 [12]. 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. More information about the CIF and its freely available CIF tool set that provides, amongst others simulation and visualization capabilities can be found at [14]. In Section 4.1, we describe the language elements of the CIF used in this paper.

In the next section, we describe the state-based tool framework, which is an instantiation of the MBE-SCT framework with tools for state-based controller synthesis.

4 Tool framework for state-based controller synthesis

Figure 4 shows the stated-based tool framework to support the supervisory controller design using the state-based supervisory controller method.. Documents, models and realizations are graphically depicted according to the convention of Figure 3. (Software) tools are represented as filled, rounded rectangles.

In the state-based tool framework, S/P_{R} and S/P_{D} represent the requirements and the design of plant P under supervision of supervisor S, and S_{R} and P_{R} denote the requirement documents of the plant and supervisor, respectively. The models used in this figures are related to the models from Figure 3 as follows: $S_{\mathrm{R}}.\mathrm{spec} \in M_{R_{\mathrm{S}}}$; $S.\mathrm{bdd}, S.\mathrm{cif} \in M_{\mathrm{S}}$; $P_{\mathrm{DE}}.\mathrm{sts}, P_{\mathrm{DE}}.\mathrm{cif}, P_{\mathrm{HY}}.\mathrm{cif} \in M_{\mathrm{P}}$; and $S/P_{\mathrm{DE}}.\mathrm{py} \in Z_{\mathrm{S}}$[3].

The control requirements S_{R} are formally modeled resulting in S_{R}.spec. Model P_{DE}.sts describes the uncontrolled discrete-event behavior of the plant using state-tree structures. The NBC tool takes as input (discrete-event) model of the uncontrolled system P_{DE}.sts and model of the control requirements S_{R}.spec. As

[3] The (file) extension '.sts' refers to a state-tree structure, and extension '.spec' refers to the specification of the control requirements, both specified using the input language for the controller synthesis software package NBC; extension '.bdd' refers to the Binary-Decision-Diagrams, one for each controllable event; extension '.cif' refers to the CIF language; and extension '.py' refers to the Python language.

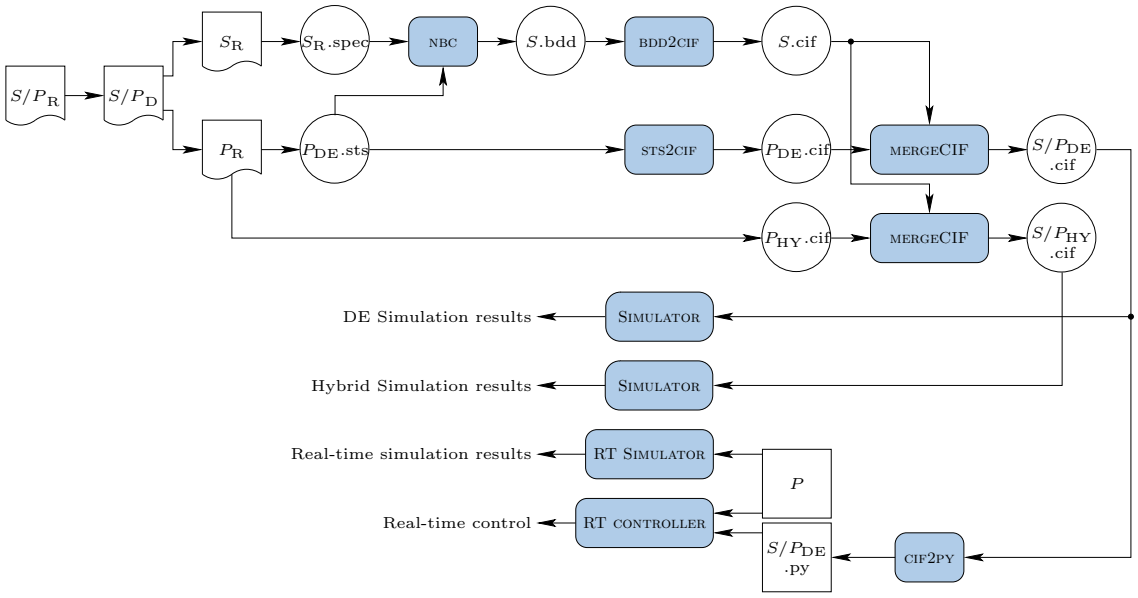

Fig. 4: Stated-based tool framework.

output, we obtain model of the supervisor S.bdd specified by BDDs, one for each controllable event.

Using the BDD2CIF translator, the model of the supervisor S.bdd is translated to an equivalent CIF model (S.cif). The STS model of the plant P_{DE}.sts can be translated to an equivalent CIF model (P_{DE}.cif) using the STS2CIF translator. 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. Using the CIF simulator (SIMULATOR), the S/P_{DE}.cif model can be simulated to analyze 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 the actual hardware of plant P. The real-time simulator (RT SIMULATOR) connects the hardware of the plant and the CIF model of the supervisor to Analise 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/P_{DE}.py that can be executed on real-time control platform RT CONTROL that is connected to the actual hardware of the plant.

The translation tools BDD2CIF, and STS2CIF, and the MERGECIF tool are formally defined in the next section. The NBC tool has been used in [8]. The other elements of the state-based tool framework are shared with the event-based tool framework from [5].

5 Formal definition of the developed translation functions

In this section, we formally define the functions STS2CIF (Section 4.1), bddcif (Section4.2), and MERGECIF (Section 4.2). These functions are the core of their respective tool implementations from the state-based tool framework.

5.1 Translation function sts2cif

First, we define the State-tree Structures (STS) and the CIF automata formally. Then, the STS2CIF translation function is defined. In this paper, we restrict the STS such that AND superstates are not allowed as child states in OR superstates. This restriction simplifies the translation to CIF considerably while a relevant subset for modeling is maintained.

Definition State-Tree Structures A State-Tree Structure (STS) (see [8] for the complete definition) is defined as $G = (\mathcal{ST}, \mathcal{H}, \Sigma, \Delta, \mathcal{ST}_0, \mathcal{ST}_m)$, where

- $\mathcal{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ is a state tree, where
  - X is a finite structured state set
  - $x_0 \in X$ is a special state called the *root state*;
  - $\mathcal{T} : X \to \{and, or, simple\}$ is the type function;
  - $\mathcal{E} : X \to 2^X$ is the nonempty expansion function.
- $\mathcal{H} = \{H^a | a \in X,\ \mathcal{T}(a) = or,\ H^a = (X^a, \Sigma^a, \delta^a, X_0^a, X_{\mathrm{m}}^a)\}$ is a set of matching holons assigned to all OR superstates of $\mathcal{ST}$;
- $\Sigma = \bigcup_{\forall_{H^a \in \mathcal{H}}} \Sigma_{\mathrm{I}}^a$ is the event set including all events appearing in $\mathcal{H}$;
- $\Delta : \mathrm{ST}(\mathcal{ST}) \times \Sigma \to \mathrm{ST}(\mathcal{ST})$, where $\mathrm{ST}(\mathcal{ST})$ denotes the set of all sub-state-trees of $\mathcal{ST}$;
- $\mathcal{ST}_0$ is the *initial sub-state tree*;
- $\mathcal{ST}_m$ is the *marker state tree set*.

A holon is defined as $H = (X^H, \Sigma^H, \delta, X_0, X_{\mathrm{m}})$, where

- X^H is the non-empty state set[4];
- Σ^H is the event set that is partitioned into the sets of controllable and uncontrollable events, i.e. $\Sigma^H = \Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{u}}$, with $\Sigma_{\mathrm{c}} \cap \Sigma_{\mathrm{u}} = \emptyset$[5];
- the transition structure $\delta : X^H \times \Sigma \rightharpoonup X^H$[6] is a partial function defining the transitions between states labeled by an event[7];

[4] In [8], the state set is structured as the disjoint union of the (possibly empty) *external stateset* X_{E} and the nonempty *internal state set* X_{I}. However, in this paper, we restrict the definitions of holons such that there are no external states.

[5] In [8], the event set can also be partitioned into a internal event set and a boundary event set. In this paper we restrict the definition of holons such that the boundary event set is empty.

[6] Notations $f : A \rightharpoonup B$ and $g : A \to B$ define a partial function f and a total function g, both with domain A and codomain B.

[7] In [8], the transition structure is partitioned into an internal transition structure and a boundary transition structure. However, in this paper, we restrict the definitions of holons such that there are no boundary transitions.

- $X_0 \subseteq X_\mathrm{I}$ is the nonempty *initial state set*;
- $X_\mathrm{m} \subseteq X_\mathrm{I}$ is the nonempty *terminal state set*.

CIF definition We assume a set of variables $\mathcal{V}$, a set of basic action labels $\mathcal{L}_\mathrm{basic}$, which does not include the predefined non-synchronizing action τ. For a set of variables $S \subseteq \mathcal{V}$, $\mathrm{Pred}(S)$ denotes the set of all predicates over variables from S, and $\mathrm{Expr}(S)$ denotes the set of all expressions over variables from S.

An *atomic interchange automaton* is a tuple (X, X_i, dtype, V, v_0, init, flow, inv, tcp, L, E) where

- $X \subseteq \mathcal{V}$ is a finite set of variables, $X_\mathrm{i} \subseteq X$ is the set of *internal* variables, and $X_\mathrm{e} = X \setminus X_\mathrm{i}$ is the set of *external* variables;
- $\mathrm{dtype} : X \to \{\mathrm{disc}, \mathrm{cont}, \mathrm{alg}\}$ is a function that associates to each variable a dynamic type. The *dynamic* type of a variable gives information about its time-dependent behavior. In this paper, we only use discrete variables (dynamic type disc). The value of a discrete variable remains constant when time passes;
- V is a finite non-empty set of *vertices*, called *locations*, and $v_0 \in V$ is the *initial location*;
- $\mathrm{init} \in \mathrm{Pred}(\tilde{X})$ is the initial condition. For $Y \subseteq X$, $\tilde{Y} = Y \cup \{\dot{y} \mid y \in Y \cap \{x | x \in X, \mathrm{dtype}(x) = \mathrm{cont}\}\}$ is the extension of Y with the dotted versions of the continuous variables in Y.
- $\mathrm{flow}, \mathrm{inv}, \mathrm{tcp} : V \to \mathrm{Pred}(\tilde{X})$, are functions that each associate to each location $v \in V$ a predicate describing the *flow condition*, the *invariant*, and the *time-can-progress predicate*, respectively;
- $L \subseteq \mathcal{L}_\mathrm{basic}$ is a finite set of synchronizing action labels;
- $E = V \times \mathrm{Pred}(\tilde{X}) \times (\mathcal{L}_\mathrm{basic} \cup \{\tau\} \cup C_X) \times (\mathcal{P}(\tilde{X}) \times \mathrm{Pred}(\tilde{X} \cup \tilde{X}^-)) \times V$ is a finite set of *edges*, such that for each element $(v, g, a, (W, r), v') \in E$, v and v' are the *source* and *target* locations, respectively, g is the *guard*, a is an *action label*, $W \subseteq \tilde{X}$ is a set of jumping variables (the value of which may change as a result of an action transition), and r is the *jump predicate*, also called *reset map*. For any $Y \subseteq (\tilde{\mathcal{V}})$, $Y^- = \{y^- \mid y \in Y\}$ denotes the set of minus superscripted variables that represent the values of variables before an action transition. Two kinds of action statements exist: basic action labels $a \in L$ that synchronize on the basis of (name) equality, and the predefined non-synchronizing τ action.[8]

The set of CIF automata $\mathcal{A}$ is defined by the following grammar for the CIF automata $\alpha \in \mathcal{A}$:

$\alpha ::= \alpha_\mathrm{atom}$ atomic interchange automaton
$\quad \mid \alpha \parallel \alpha$ parallel composition

[8] The full-blown CIF framework also contains a CSP-based communication mechanism to communicate values over channels that connect two parallel automata. However, in this paper we do not need this.

Translation function sts2cif The STS2CIF translation function translates a state-tree structure to a CIF automaton. Intuitively, an AND superstate is translated to a parallel composition of the respective translations of its child states; an OR superstate is translated to an atomic interchange automaton, with a location for each child state and edges between the locations as specified in the holon associated with the OR superstate; and a simple state is translated to an atomic interchange automaton with a single mode without edges. Formally, function STS2CIF is defined as follows. Let $G = (X, x_0, \mathcal{T}, \mathcal{E}), \mathcal{H}, \Sigma, \Delta, \mathcal{ST}_0, \mathcal{ST}_m)$ be a STS, then $\text{STS2CIF}(G) = \|_{\forall a:\ a \in \text{ST2CIF}((X,x_0,\mathcal{T},\mathcal{E}),\mathcal{H})} a$, where

$$\text{ST2CIF}((X, x_0, \mathcal{T}, \mathcal{E}), \mathcal{H}) = \begin{cases} \{(\emptyset, \emptyset, \Theta, \{\mathcal{E}(x_0)\}, \mathcal{E}(x_0), \text{true}, \theta, \theta, \theta, \emptyset, \emptyset)\} & \text{if } \mathcal{T}(x_0) = \textit{simple} \\ \bigcup_{x:\ x \in \mathcal{E}(x_0)} \{\text{HOLON2ATAUT}(\mathcal{H}(x))\} & \text{if } \mathcal{T}(x_0) = \textit{or} \\ \bigcup_{x:\ x \in \mathcal{E}(x_0)} \{\text{ST2CIF}((X, x, \mathcal{T}, \mathcal{E}), \mathcal{H})\} & \text{if } \mathcal{T}(x_0) = \textit{and} \end{cases}$$

where θ denotes the function $\text{dom}(\theta) = x_0, \theta(x_0) = \text{true}$, Θ denotes a function with an empty domain, and function HOLON2ATAUT is defined as follows. Let H be a holon $H = (X^H, \Sigma^H, \delta^H, X_0^H, X_\text{m}^H)$. Then $\text{HOLON2ATAUT}(H) = (X,\ X_\text{i},$ dtype, $V,\ v_0,$ init, flow, inv, tcp, $L,\ E)$, where

- $X = \{'cs\_' +\!\!+ \text{Id}(H)\}$
- $X_\text{i} = \emptyset$
- $\text{dom}(\text{dtype}) = X, \forall_{x \in X}\ \text{dtype}(x) = \text{disc}$
- $V = X^H$
- $v_0 \in X_0^H$
- $\text{init} = 'cs\_' +\!\!+ \text{Id}(H) = v_0$
- $\text{dom}(\text{flow}) = V, \forall_{v:\ v \in V} \text{flow}(v) = \text{true}$
- $\text{dom}(\text{inv}) = V, \forall_{v:\ v \in V} \text{inv}(v) = \text{true}$
- $\text{dom}(\text{tcp}) = V, \forall_{v:\ v \in V} \text{tcp}(v) = \text{true}$
- $L = \Sigma^H$
- $E = \{(v, \text{true}, a, (\{'cs\_' +\!\!+ \text{Id}(H)\}, 'cs\_' +\!\!+ \text{Id}(H) = v'), v') | (v, a, v') \in \delta^H\}$

where function Id assigns a unique name to a holon, and $+\!\!+$ denotes string concatenation. The discrete variable $'cs\_' +\!\!+ \text{Id}(H)\}$ is used to keep track of the current state of holon H.

5.2 Translation function bdd2cif

From the State-based Supervisory Synthesis, we obtain for each controllable event a BDD. Such BDD describes whether or not its controllable event is enabled, given the current state of the plant. First, we describe the BDD formally (Section 4.2). Then, the translation function BDD2CIF that translates the BDDs to a single CIF automaton is described (Section 4.2).

Binary Decision Diagram Let $G = (\mathcal{ST}, \mathcal{H}, \Sigma, \Delta, \mathcal{ST}_0, \mathcal{ST}_m)$ be a STS, then the BDDs that are obtained from the supervisory controller synthesis are defined as follows.

A *binary decision diagram* (BDD) is defined as (N, n_init, M, E), where

- N is the finite nonempty set of nodes, including the terminal nodes n_{true} and n_{false}; $n_{\text{init}} \in N$ is the initial node;
- $M : N \setminus \{n_{\text{true}}, n_{\text{false}}\} \to \{\text{Id}(H) | H \in \mathcal{H}\} \times \mathbb{N}$ labels each node with a pair (h, i), where h denotes a holon and i denotes the i-th decision variable from h. For each node n, a pair (h_n, i_n) partitions the set of states from h_n.
- Given a node, function $E : N \setminus \{n_{\text{true}}, n_{\text{false}}\} \to N \times N$ returns the pair of nodes, where the first node is the target node of the 'false' edge, and the second node is the target node of the 'true' edge.

Translation function bdd2cif Translation function BDD2CIF translates the BDDs to a single CIF automaton consisting of one location and a self-loop for each controllable event. Such self-loop is labeled with a guard which is the BDD of the controllable event encoded as a predicate; and the controllable event itself. Formally, function BDD2CIF is defined as follows. Let bddfuncs denote the function from controllable event to a BDD that is obtained by the controller synthesis. Then, BDD2CIF(bddfuncs) = (X, X_{i}, dtype, $\{v_0\}$, v_0, true, flow, inv, tcp, L, E), where

- $X = csvars$
- $X_{\text{i}} = \emptyset$
- $\text{dom}(\text{dtype}) = X, \forall_{x \in X}\ \text{dtype}(x) = \text{disc}$
- $\text{dom}(\text{flow}) = v_0, \text{flow}(v_0) = \text{true}$
- $\text{dom}(\text{inv}) = v_0, \text{inv}(v_0) = \text{true}$
- $\text{dom}(\text{tcp}) = v_0, \text{tcp}(v_0) = \text{true}$
- $L = \text{dom}(\text{bddfuncs})$
- $E = \{(v_0, \text{EVALBDD}_a(csvars), a, (\emptyset, \text{true}), v_0) | a \in \text{dom}(\text{bddfuncs})\}$

where $csvars = \bigcup_{(N, n_{\text{init}}, M, E) \in \text{codomain}(\text{bddfuncs})}\ \bigcup_{n:\ n \in \text{dom}(M)}\ \{'cs\_' + M(n).0\}$. The variables from $csvars$ are used to describe the current state of the STS. For each controllable event a there is a function EVALBDD_a that encodes the BDD bddfuncs(a) in terms of variables $csvars$. Given the valuation of the variables from $csvars$, it returns true if controllable event a is enabled, and false otherwise.

5.3 Function mergecif

Function MERGECIF : $\alpha \times \alpha \to \alpha$, where α denotes the set of CIF automata, takes two CIF automata and returns the parallel composition of them. MERGECIF$(a, b) = a \parallel b$. In the parallel composition, automata a and b share the external variables $X_{e_a} \cap X_{e_b}$, and synchronize on the actions $L_a \cap L_b$.

Function MERGECIF is used to merge the CIF automaton obtained by translating the STS description of the plant (observer) and the CIF automaton obtained by translating the BDDs (supervisor). Note that restrictions on controllable events in the observer are not part of the BDDs, but they are taken into account in the parallel composition of the observer and the supervisor.

6 Case study: Patient Support System

An MRI scanner, see Figure 5, is used in medical 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 6), a user interface and a light-visor is used.

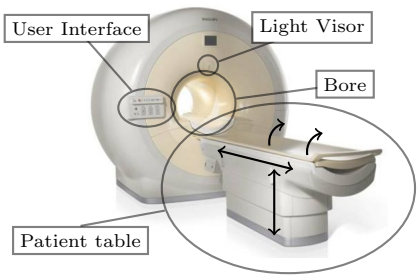

Fig. 5: MRI scanner

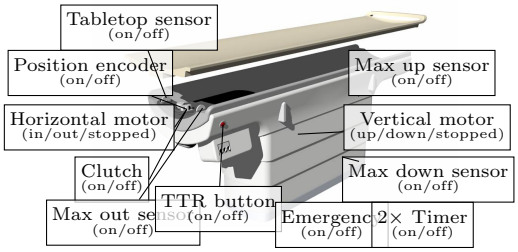

Fig. 6: Patient table

The patient support table 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 a hardware 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.

6.1 Controller synthesis

Models of the uncontrolled plant The STS model of the uncontrolled plant (see Figure 7) consists a root AND superstate *PSS* containing the AND superstates *Vertical*, *Horizontal*, *PICU*, *Light Visor* and *Emergency*. AND superstate *vertical* consists of three OR superstates modeling the sensors(*VSensors*), the motor (*VMotor*), and the relation between the motor and the sensors (*VRelation*);

The STS model of the vertical axis including the holons of its childs is shown in Figure 8[9].

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

[9] In the figure, solid and dashed edges denote controllable and uncontrollable events, respectively.

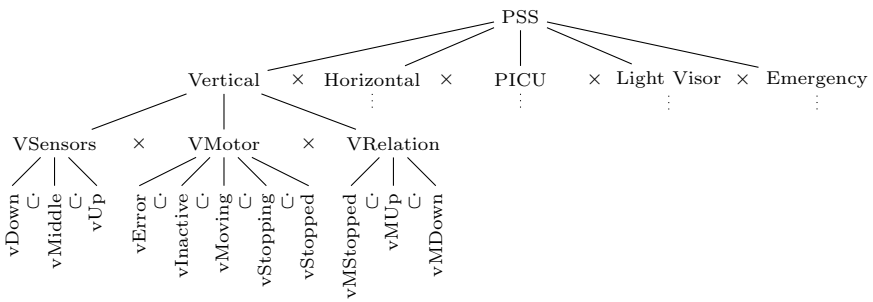

Fig. 7: State tree of the STS model.

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 (OR state VSensor). Because of the physical location, the sensors are never active at the same time.

The motor is initially in an error state (OR state VMotor). 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 *v_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.

The sensors do not change state when the motor is not moving (OR state VRelation). 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.

Models of the requirements The control requirements are described by using logical expressions, similar to the ones given in Section 2. The requirement "the vertical axis should not move beyond its maximally up and maximally down position" is modeled with the following specifications. In the maximally up position it is not allowed to move up $\{\text{VUP}\} \nrightarrow v\_move\_up$, in the maximally down position it is not allowed to move down $\{\text{VDOWN}\} \nrightarrow v\_move\_down$. When the table is maximally up or down, it should stop, otherwise the stop events must be disabled $\{\text{VDOWN}, \text{VMID}\} \nrightarrow v\_stop\_up$, $\{\text{VUP}, \text{VMID}\} \nrightarrow v\_stop\_down$.

Synthesis of the supervisor Using the NBC tool, the supervisor can be synthesized in a few seconds using an average desktop computer.

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

Figure 9 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

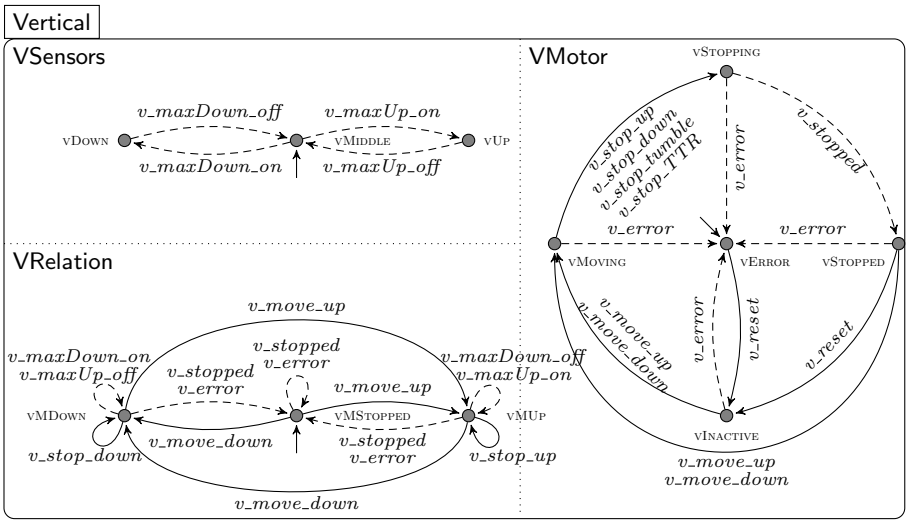

Fig. 8: Plant model of the vertical axis ($\in P_{\text{DE}}$.sts).

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 outerside (innerside) denotes an output (input) variable. Locations are visualized by means of circles labeled with the name of the location. 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 Motor modes equals $\dot{position} = speed$. The automata Motor and MotorStop synchronize on the *stop* event. The automata Motor and Sensors share the continuous variable *position*.

6.3 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

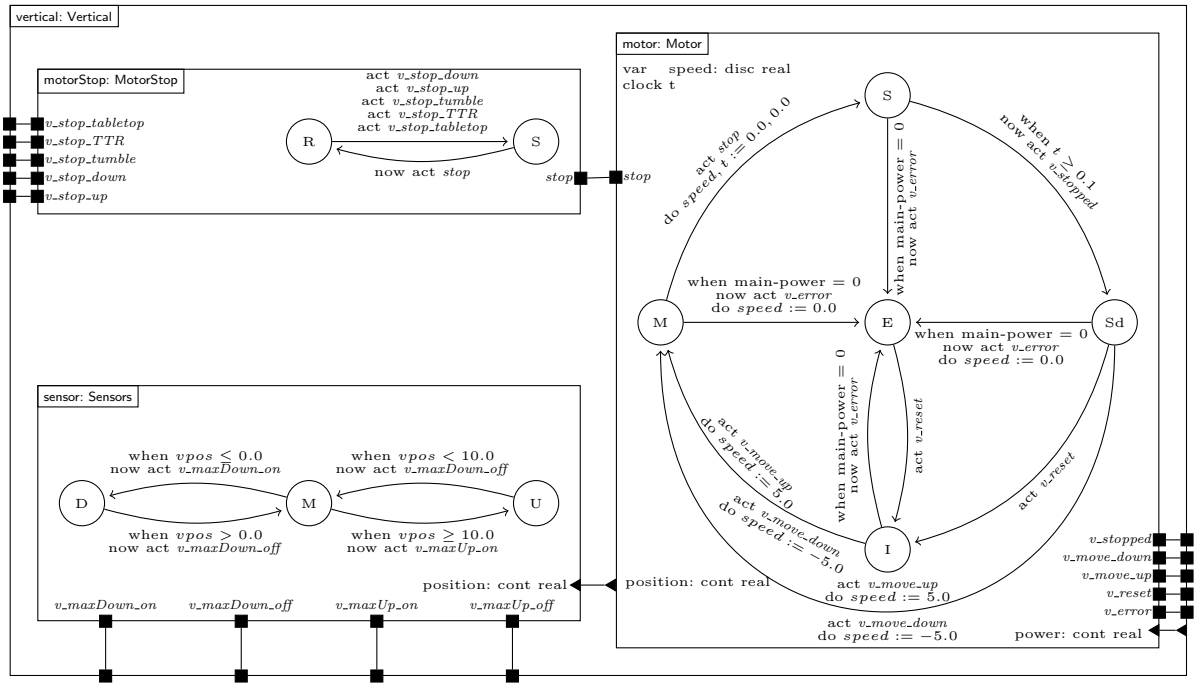

Fig. 9: Hybrid CIF model of the vertical axis ($\in P_{\text{HY}}$.cif).

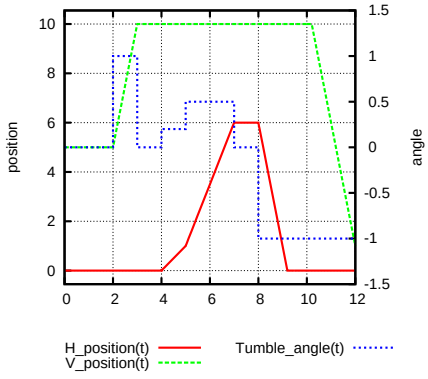

Fig. 10: Simulation results S/P_{HY}.cif.

Simulation results S/P_{HY}.cif Figure 10 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 ($t \approx 10$) until it reaches the lowest position.

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.

7 Concluding remarks

In this paper, we developed a (tool) framework for *State-based supervisory controller synthesis*. Using the Common Interchange Formalism for hybrid systems (CIF), it integrates the Model-Based Engineering and Supervisory Control Theory paradigms. The framework has been successfully used to design a supervisory controller for the patient support system of an MRI scanner. In future work, as part of the MULTIFORM project, the CIF will be extended with AND/OR superstates.

References

1. Patton, R.J., Frank, P.M., Clarke, R.N., eds.: Fault diagnosis in dynamic systems: theory and application. Prentice-Hall, Inc. (1989)
2. Ogren, I.: On the principles for model-based systems engineering. Systems Engineering **3**(1) (2000) 38–49
3. Braspenning, N.C.W.M.: Model-Based Integration and Testing of High-Tech Multidisciplinary Systems. PhD thesis, Eindhoven University of Technology (2008)
4. Wonham, W.: Supervisory control of discrete-event systems. Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada (2007)
5. Schiffelers, R.R.H., Theunissen, R.J.M., Beek, D.A.v., Rooda, J.E.: Model-based engineering of supervisory controllers using cif. In: 3rd IFAC Conference on Analysis and Design of Hybrid Systems. (2009) submitted for review
6. Beek, D.A.v., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Foundations of an interchange format for hybrid systems. In Bemporad, A., Bicchi, A., Butazzo, G., eds.: Hybrid Systems: Computation and Control, 10th International Workshop. Volume 4416 of Lecture Notes in Computer Science., Pisa, Springer-Verlag (2007) 587–600
7. Beek, D.A.v., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Revised hybrid system interchange format. Technical Report HYCON Deliverable D3.6.3, HYCON NoE (2007)
8. Ma, C., Wonham, W.: Nonblocking supervisory control of state tree structures. Volume 317/2005 of Lecture Notes in Control and Information Sciences. Springer (2005)
9. Ma, C., Wonham, W.: Nonblocking supervisory control of state tree structures. Automatic Control, IEEE Transactions on **51**(5) (2006) 782–793
10. Beek, D.A.v., Reniers, M., Rooda, J.E., Schiffelers, R.R.H.: Concrete syntax and semantics of the compositional interchange format for hybrid systems. In: 17th Triennial World Congress of the International Federation of Automatic Control, Seoul, Korea (2008)
11. HYCON Network of Excellence: http://www.ist-hycon.org/ (2005)
12. MULTIFORM consortium: Integrated multi-formalism tool support for the design of networked embedded control systems MULTIFORM. http://www.multiform.bci.tu-dortmund.de (2008)
13. Sonntag, C., Schiffelers, R.R.H., Beek, D.A.v., Rooda, J.E., Engell, S.: Modeling and simulation using the Compositional Interchange Format for hybrid systems. In Troch, I., Breitenecker, F., eds.: 6th International Conference on Mathematical Modelling, Vienna, Austria (2009)

14. Systems Engineering Group TU/e: CIF toolset. http://se.wtb.tue.nl/sewiki/cif (2008)