Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner

R.J.M. Theunissen, M. Petreczky, R.R.H. Schiffelers, D.A. van Beek, J.E. Rooda

***Abstract*—In this paper we present a case-study on application of Ramadge-Wonham supervisory control theory (abbreviated by SCT in the sequel) to a patient support system of a magnetic resonance imaging (MRI) scanner. We discuss the whole developmental cycle, starting from the mathematical models of the uncontrolled system and of the control requirements, and ending with the implementation of the obtained controller on the actual hardware. The obtained controller was tested on the physical system. In this case study, we attempted to build the models in a modular way, in order to decrease the computational complexity of the controller synthesis and to improve the adaptability of the models. An important advantage of SCT is that it allows automatic generation of the controller, and that it can thus improve adaptability of the control software. We also briefly discuss our experience on the adaptability of the control software, obtained in the course of this case study.**

Note to Practitioners **Current industrial practice of supervisory controller design is based on a separation between informal specification of behavioral requirements by domain experts, and encoding of these requirements in control software by software specialists, leading to code and requirements that are difficult to develop, debug, maintain, and adapt. We propose a supervisory controller design process that instead relies on modeling the behavioral requirements and uncontrolled system, and generating the controller by means of supervisory control synthesis.**

Where supervisory control synthesis provides technology to develop the controller right, we employ simulation-based validation to ensure that the right controller is built. This can be done by means of execution of user-defined scenarios, and generation of graphs showing the evolution of the model variables as a function of time. The simulator also supports real-time, interactive, simulation and animation, based on user supplied images of the system in the standardized SVG (Scalable Vector Graphics) format.

The discussed new development process of supervisory controllers has been demonstrated to be highly effective for generation of code, that has been used for real-time control of an actual patient support table of an MRI system. The enormous potential for reduction of development time of new controller functionality is illustrated by means of an actual user modification request case.

The new supervisory controller development process is currently under investigation by several high tech industries in Eindhoven. A new project has been defined together with several high tech companies, including an innovative software company, to further develop the methods, techniques and tools to a level that is suited for commercial application.

***Index Terms*—Supervisory control synthesis, modeling, simulation, validation, real-time control.**

This work has been carried out as part of the DARWIN project at Philips Healthcare under the responsibilities of the Embedded Systems Institute (ESI). This project was partially supported by the Dutch Ministry of Economics Affairs under the BSIK program.

R.J.M. Theunissen is with Nspyre, Herculesplein 24, Utrecht, Netherlands.

M. Petreczky is with the Dept. Computer Science and Automatic Control, Ecole des Mines de Douai, France.

R.R.H. Schiffelers is with ASML, De Run 6501, Veldhoven, Netherlands.

D.A. van Beek and J.E. Rooda are with the Dept. Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513, Eindhoven, Netherlands.

I. Introduction

The goal of this paper is to present a specific application of the Ramadge-Wonham supervisory control theory (SCT), see [1], [2], to control problems arising in mechatronic systems. The application addresses the control of a patient support table for MRI scanners of Philips Medical Systems.

Contribution of the paper: The paper describes the main steps of the control synthesis for the application described above. First, a discrete-event (finite-state automaton) model of the uncontrolled patient support table, which is referred to as the *plant* in supervisory control terminology, was created. Only the behavior of the plant in the absence of errors is modeled. Second, the control requirements were modeled as automata. The language accepted by this automaton represents those sequences of events which the closed-loop system is allowed to generate. Then, SCT is applied to obtain a supervisor and the supervisor is used to generate the desired controller. The obtained controller was first tested by means of simulation, and then it was *implemented and tested on a MRI scanner*.

It is worth noting that not only is a supervisory controller generated for the particular case-study, but also a tool chain is developed which allows the automatic translation of the obtained supervisor to a real-time controller. This tool chain may serve as a starting point for a general purpose software tool for generating real-time controllers based on SCT.

Finally, note that the models presented in this paper are slight modifications of the models which were used to synthesize the supervisor that was implemented on existing hardware. These modifications were applied to render the models easier to understand. However, they do not influence the functional behavior of the models. Note that software simulation reveals that the supervisor obtained from the models of this paper also satisfies the control requirements.

Motivation for SCT: The control problem described in this paper can also be solved using other techniques. Indeed, a solution that was developed manually, already existed prior to the SCT solution. The reason why we chose SCT for this case study, is that SCT naturally supports easy adaptation of the control software. In the sequel, we use the word 'evolvability' to denote the property that the controller can

be easily modified to accomodate changed control objectives and/or changed plants. From the point of view of evolvability, the most attractive feature of SCT is that it allows automatic generation of a controller from the formal model of the plant and the control requirements, such that the generated controller is guaranteed to meet those control requirements. That is, instead of modifying the existing controller, we modify the model of the control requirements and automatically generate a new controller, which is guaranteed to meet the modified control requirements.

For many mechatronic applications, including the one described in this paper, evolvability is one of the prime requirements. For this case study, evolvability was tested by redesigning it for slightly different control requirements; the design and implementation (on an actual MRI scanner) of the new controller took only half a day, see Section V-A for more details. This encouraging result lead us to conjecture that SCT can be useful for improving evolvability of controllers. It would be interesting to perform a thorough comparison of evolvability properties of various SCT and non-SCT based methods, however such a comparison is beyond the scope of this paper.

Related work: Despite the fact that SCT is well established, the number of industrial applications is limited. Without claiming completeness, previous applications of SCT include: a rapid thermal multiprocessor [3], mobile robots [4], passenger land-transport systems [5], a water bath boiler [6], under-load tap-changing transformers [7], and automated manufacturing and assembly systems: [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. To the best of our knowledge, the application domain of MRI scanners is new. Some of the results of the paper were announced in the conference proceedings [19]. The main difference with the previous paper [19], is that the current paper presents more detailed models and a modeling methodology. In addition, it systematically explains the role of supervisory control in improving evolvability. A preliminary version of the paper appeared as a technical report [20]. In [21], an application of state-based SCT to a patient communication system of MRI scanners was explored. The main difference with respect to [21] is that we consider the patient support table, and that we used the event based version of SCT. Moreover, unlike [21], we also tested the controller on real hardware. Related work on real-time implementations of supervisory controllers is discussed in Section VI-D.

II. Functionality of the patient support system of an MRI scanner

The patient support system is used to position a patient in an MRI scanner, see Figure 1. An MRI scanner is used mainly in medical diagnosis to render pictures of the inside of a patient non-invasively. The patient support system (Figure 2) 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

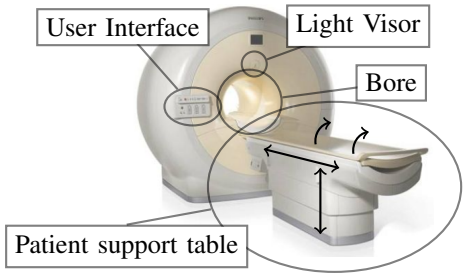


Fig. 1. MRI scanner

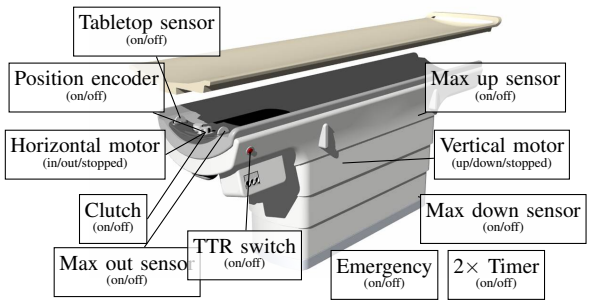


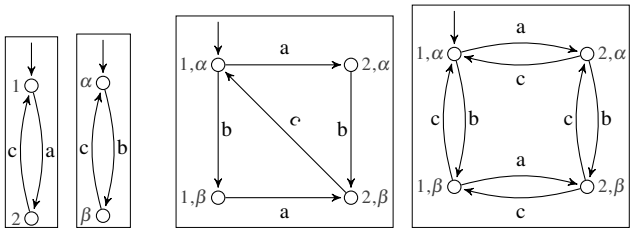
Fig. 2. Patient table

of a motor drive depending on the state of the clutch. It contains sensors to detect both the presence of the tabletop and its position. The system is equipped with hardware safety systems, namely an emergency stop and the tabletop release switch TTR, that allow the operator to override the control system in emergency situations. The system is controlled via a user-interface UI. This interface contains a tumble-switch to control the movement of the table, and three buttons to control the clutch, the emergency system and the light-visor with automatic positioning. Furthermore, the UI contains LEDs to display the current state of the system to the operator.

The supervisor should accomplish multiple control objectives. When the operator operates the tumble-switch, the table should move up and down, or in and out of the bore. This depends on the current position of the table and the position of the tumble-switch. When the manual button is pushed, the clutch should be released such that the table can be moved manually by hand. Finally, the table should not move beyond its end positions, and it should not collide with the magnet.

The patient support system is more difficult to control than might appear at first sight. It contains several complex interactions of components. Even though the version of the patient support system that is discussed in this article is simplified compared to the real system, the control requirements consist of 62720 reachable states, and 869520 transitions. Another parameter which reflects the complexity is that the plant model together with the model of the control requirements, as discussed in this article, consists of 31 automata of 2 to 4 states, and one automaton with 7 states.

The complexity of the designer's task becomes even more apparent when one considers the time which is required to build the control software manually. In fact, it was estimated that one would need a week for manual adaptation of the control software to meet the modified requirements described in Section V-A. Note that with the approach of this paper,



(a) A (b) B (c) A ‖ B (d) A ||| B

Fig. 3. synchronous and interleaving parallel composition

the adaptation of the models of control requirements and the generation of new control software took merely half a day.

III. Model structure

The goal of this section is to sketch the structure of the models of the plant and of the control requirements. The detailed models can be found in Section IV. In this paper, we will use the classical Ramadge-Wonham framework [1] for modeling the plant, the supervisor and the requirements. In particular, we will tacitly use the terminology from [1].

We model the plant and the control requirements in a modular way, i.e. we represent them as parallel interconnection of several components. In turn, each component is modeled by a finite-state automaton. In order to compose finite-state automata, we will use two kinds of parallel compositions: 1) *synchronizing parallel composition*, see [1], which requires synchronous execution of shared events (events with common labels) and interleaved (independent) execution otherwise; and 2) *interleaving parallel composition*, see [22], which defines interleaved execution for all events. In this article, synchronous parallel composition is denoted by ‖, and interleaving parallel composition is denoted by |||. Figure 3 shows an example for each of the two parallel composition operators. Note that interleaving parallel composition can result in nondeterministic models. Since the classical Ramadge-Wonham framework used in this paper cannot handle nondeterministic systems, this could in principle lead to problems. However, in our case, interleaving does not result in a nondeterministic model. The events which are shared by the automata which are composed by interleaving parallel composition, only occur in self-loops.

A. Plant model

The plant is represented as a parallel interconnection of automata representing the following plant components:

- Model of the vertical axis VAxis,
- Model of the horizontal axis HAxis,
- Model of the user interface UI, which describes the effect of the actions of the operator on the plant.

The model of the complete uncontrolled patient support system is thus defined as the following synchronous parallel composition:

$$\mathsf{VAxis} \parallel \mathsf{HAxis} \parallel \mathsf{UI}$$

Each component above represents a separate aspect of the plant's functionality. The components themselves are modeled as a parallel interconnection of the models of the: *actuators*, the *sensors*, and the *relations between the actuators and sensors*. The actuator-sensor relations represent the physical structure of the machine, relating actions of actuators to activations of sensors.

Thus for example, the model of the vertical axis is of the form

$$\mathsf{VAxis} \triangleq \mathsf{VActuators} \parallel \mathsf{VSensors} \parallel \mathsf{VRelations}$$

Here, VActuators describes the model of the motor which moves the vertical axis. It essentially defines that the motor can be started and then stopped. The automaton VSensors models the functioning of the sensors which detect if the table is in the maximally down or up position. It defines that a sensor can be switched on when it is off, and vice versa. Finally, VRelations models the actuator-sensor relation: as a result of their physical positions, the vertical sensors cannot be active at the same time. Furthermore, the sensors cannot change their state if the motor is not moving. In somewhat more detail: the maximally up sensor may be activated only when the table is moving up, and the maximally down sensor may be activated only when the table is moving down.

The detailed description of the vertical axis model as well as the models of the other components can be found in Section IV.

B. Control requirements

The control requirements of interest are *safety* requirements. More precisely, the control requirements are sets of sequences of events which the closed-loop system is allowed to generate. The control requirements are represented by finite-state automata. The language accepted by the automaton is exactly the set of all *safe* sequences of events which the closed-loop system is allowed to generate.

The components of the control requirements reflect the components of the plant. That is, for each plant component we model the corresponding control requirement separately. Hence, the model of the control requirements is of the form

$$\mathsf{VReq} \parallel \mathsf{HReq} \parallel \mathsf{HVReq} \parallel \mathsf{UIReq}$$

Here, VReq is the model of the control requirements for the vertical axis, HReq is the model of the control requirements for the horizontal axis, HVReq is the model of control requirements pertaining to the interaction between the horizontal and vertical axis, and finally UIReq is the model of the control requirements for the interface. For example, VReq formalizes the following requirements. Movement beyond the maximally up position is not allowed. This implies that initiating movement in the upper direction must be prevented when the table is maximally up. Furthermore, movement in the upper direction must be stopped when the table reaches the maximally up position. The detailed models can be found in Section IV.

IV. Models

Below we present the formal models for the plants and control requirements. The presentation is done component-wise. That is, we first present the plant model and the model

of control requirements for the vertical axis, then for the horizontal one, then for the interaction of the axes, and finally we present the plant model and the model of control requirements for the user interface.

In the models, states are denoted by vertices, initial states are indicated by an unconnected incoming arrow, and marked states are denoted by filled vertices. Some of the models use both marked and unmarked states, e.g. Figures 4, 8a, 14a. In such models, the marked states represent stable states, that should always be reachable, whereas the unmarked states represent transient, temporary states, that should eventually exit to a marked state. The non-blocking property of supervisory control synthesis algorithms ensures that marking states are indeed always reachable. For more details on marked states and on the non-blocking property see [1].

Controllable and uncontrollable events are depicted by solid and dashed edges, respectively. If several event names are indicated on an edge, then this should be understood as a collection of edges (with the same source and target as the depicted edge), with each edge labelled by one of the events. A bold event name indicates a set of events, representing an edge for each event in the set.

The control requirements are divided into sub-requirements that are each modeled by means of small, independent, automata. To this end, we have to introduce additional events. In particular by introduction of the internal event *normal*, see Figure 14a, and by splitting the horizontal and vertical stop invents, ***vStop*** and ***hStop***, respectively, into different sub events, see Section IV-A1 and IV-B1.

A. Vertical axis

The patient table can move up and down along the vertical axis. The vertical axis contains two end sensors, maximally up and maximally down, and one actuator for the vertical motor drive. The motor can move the table up and down. The system should never move beyond the maximally up and down position.

1) Plant model (VAxis): The plant model of the vertical axis consists of the synchronous parallel composition of the models of the actuators, sensors and structure of the vertical axis:

$$\mathsf{VAxis} \triangleq \mathsf{VActuators} \parallel \mathsf{VSensors} \parallel \mathsf{VRelations}$$

The actuator, sensor and structure models are defined as:

$$\mathsf{VActuators} \triangleq \mathsf{VMotor}$$
$$\mathsf{VSensors} \triangleq \mathsf{VUpSensor} \parallel \mathsf{VDownSensor}$$
$$\mathsf{VRelations} \triangleq \mathsf{VSensorsRelation} \parallel \mathsf{VMotorSensorRelation}$$

a) Motor drive (VMotor): The motor is controlled by a resource controller, which controls the brakes, and calculates set-points for the feedback control loop. The model of the motor drive only includes the behavior exposed to the supervisor, see Figure 4. Initially, the motor is stopped. From this state, a movement can be started. The event set ***vMove*** $\triangleq$ {*vMoveUp*, *vMoveDown*} is an abbreviation for two move events. If the motor is moving and a stop event in the set

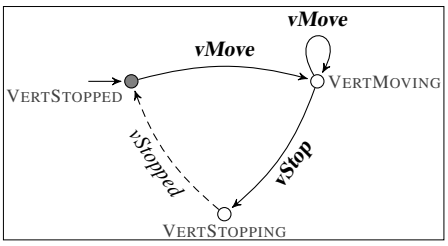


Fig. 4. Model VActuators: vertical actuators

(a) VDownSensor (b) VUpSensor

Fig. 5. Models VSensors: vertical sensors

vStop $\triangleq$ {*vStopUp*, *vStopDown*, *vStopTTR*, *vStopTumble*}, is triggered, the motor slows down to come to a halt. When the motor has come to a halt, the event *vStopped* is emitted, and the motor enters the stopped state again. Only the stopped state is marked, because the motor must always be able to return to the stopped state.

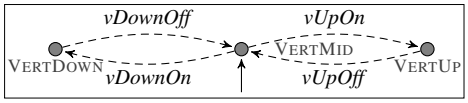
Distinguishing different stop events facilitates decomposition of the complete stop behavior into multiple independent requirements. Individual stop events should be enabled in distinct cases. For instance, *vStopUp* is enabled only when the table has reached its maximally up position. By having a different stop event for each case, these stop events do not synchronize, so that the cases can be modeled independently of one another. The stop events *vStopUp* and *vStopDown* are used in the control requirements of Figures 7a–7c, whereas the stop events *vStopTTR* and *vStopTumble* are used in the control requirements of Figures 14a and 16a, respectively.

b) Sensors (VDownSensor, VUpSensor): The maximally down and maximally up sensors are modeled in Figures 5a and 5b. The sensors are active if the table is at the sensor position, otherwise the sensors are inactive. This is modeled by means of two (marked) states, namely ON and OFF. The sensors emit the uncontrollable events *vDownOn* (*vUpOn*) or *vDownOff* (*vUpOff*), when a sensor becomes active or ceases to be active, respectively. Initially the table is assumed to be neither up or down, so that both end sensors are inactive, indicated by the states VERTDOWNOFF and VERTUPOFF.

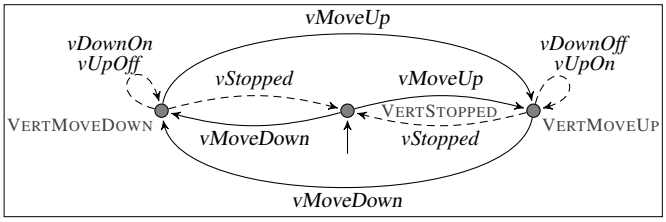
c) Sensor-sensor relation (VSensorsRelation): The two sensors are never active at the same time, as a result of their physical location. This relation is modeled in Figure 6a. The model includes the complete behavior of the two individual sensors.

d) Motor-sensor relation (VMotorSensorRelation): The sensors do not change state when the table is not moving vertically, see Figure 6b. Only when the motor drive is moving the table up, the maximally down sensor can turn off (*vDownOff*) and the maximally up sensor can turn on (*vUpOn*), and likewise for the opposite direction.

2) Control requirements (VReq): The requirement model of the vertical axis consists of the synchronous parallel com-

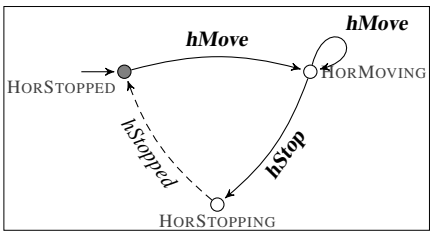


(a) VSensorsRelation

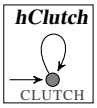


(b) VMotorSensorRelation

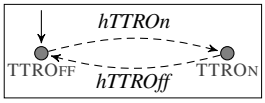
Fig. 6. Models VRelations: vertical structure



(a) HMotor

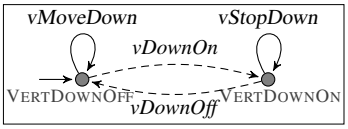


(b) HClutch

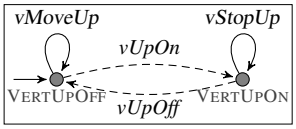


(c) HTTRSwitch

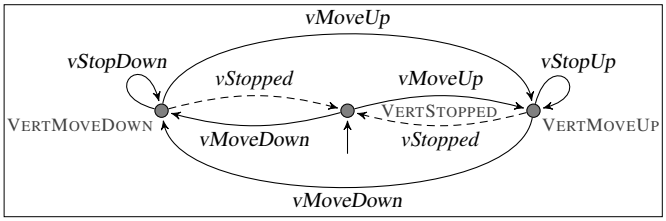
Fig. 8. Model HActuators: horizontal actuators



(a) VStopDown



(b) VStopUp



(c) VStopUpDown

Fig. 7. Model VReq: vertical axis requirements

position of multiple control requirements, namely:

$$\mathsf{VReq} \triangleq \mathsf{VStopDown} \parallel \mathsf{VStopUp} \parallel \mathsf{VStopUpDown}$$

a) Maximally up and down (VStopDown, VStopUp, VStopUpDown): Movement beyond the maximally up position is not allowed. This implies that initiating movement in the upper direction must be prevented when the table is maximally up. Furthermore, movement in the upper direction must be stopped when the table reaches the maximally up position. Likewise it is not allowed to move beyond the maximally down position.

These requirements are modeled in Figures 7a–7c. First, the event *vMoveDown* (*vMoveUp*) is only allowed if the table is not maximally down (up). Second, the event *vStopDown* (*vStopUp*) is enabled only in the maximally down (up) position. This allows the table to stop when the end position has been reached. Finally, the *vStopDown* (*vStopUp*) event is enabled only if the motor is moving down (up), see Figure 7c. The synchronizing semantics of the parallel composition in VStopDown ∥ VStopUp ∥ VStopUpDown ensures that event *vStopDown* (*vStopUp*) is enabled only if the states VERTDOWNON (VERTUPON) and VERTMOVEDOWN (VERTMOVEUP) are both active (see Figures 7a, 7b, and 7c).

B. Horizontal axis

The horizontal axis consists of a removable tabletop on top of the main table. The tabletop (if present) can be moved in and out of the bore. It can be added and removed only in the maximally out position. The presence of the tabletop is detected by a sensor. Like the vertical axis, the horizontal axis contains two end sensors, maximally in and maximally out, and a motor drive. The motor drive is coupled to the tabletop by a clutch. When the clutch is released, the tabletop (if present) can be moved freely by an operator, otherwise the positioning is controlled through the motor drive. Finally, the control of the clutch can be overridden by a hardware safety system, called Table Top Release (TTR). The clutch is released when the TTR switch is active, independently of the controller. The system should never move beyond the maximally in and maximally out positions. Furthermore, the horizontal axis may not move, when the tabletop is not present, when the clutch is released or when the TTR switch is active.

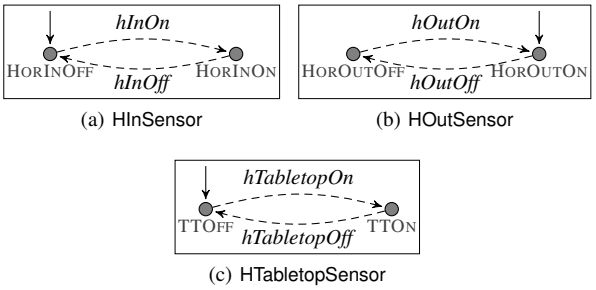
1) Plant model (HAxis): The plant model of the horizontal axis consists of the synchronous parallel composition of the models of the actuators, sensors and structure of the horizontal axis:

$$\mathsf{HAxis} \triangleq \mathsf{HActuators} \parallel \mathsf{HSensors} \parallel \mathsf{HRelations}$$

The actuator, sensor and structure models are defined as:

$$\mathsf{HActuators} \triangleq \mathsf{HMotor} \parallel \mathsf{HClutch} \parallel \mathsf{HTTRSwitch}$$
$$\mathsf{HSensors} \triangleq \mathsf{HInSensor} \parallel \mathsf{HOutSensor} \parallel \mathsf{HTabletopSensor}$$
$$\mathsf{HRelations} \triangleq \mathsf{HSensorsRelation} \parallel \mathsf{HActuatorSensorRelations}$$

a) Actuators (HMotor, HClutch, HTTRSwitch): The horizontal motor drive is similar to the vertical motor drive, see Figure 8a. The event set $\boldsymbol{hStop} \triangleq \{hStopIn, hStopOut, hStopTTR, hStopTabletop, hStopTumble\}$ denotes a number of different stop events. The event set $\boldsymbol{hClutch} \triangleq \{hClutchOn, hClutchOff\}$ is an abbreviation for the clutch events, and the event set $\boldsymbol{hMove} \triangleq \{hMoveIn, hMoveOutNormal, hMoveOutRestricted\}$ is an abbreviation for the move events. The clutch is modeled with one state (CLUTCH) in which the clutch events are self-looped, see Figure 8b. The tabletop release switch is modeled similarly to a sensor, see Figure 8c.



(a) HInSensor (b) HOutSensor

(c) HTabletopSensor

Fig. 9. Model HSensors: horizontal sensors

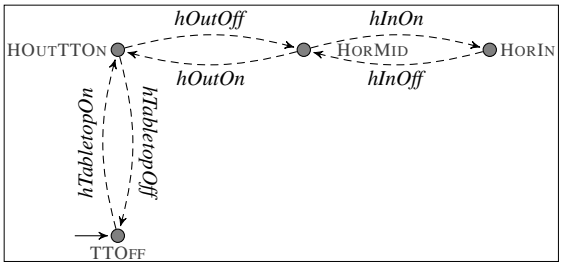


Fig. 10. Model HSensorsRelation: horizontal structure (synchronizing parallel composition part)

b) Sensors (HInSensor, HOutSensor, HTabletopSensor): The sensors of the horizontal axis are modeled similarly to the sensors in the vertical axis, see Figures 9a–(c). Initially, the tabletop is not present (TTOFF), the maximally out sensor is on (HOUTON), and the maximally in sensor is off (VMAXINOFF).

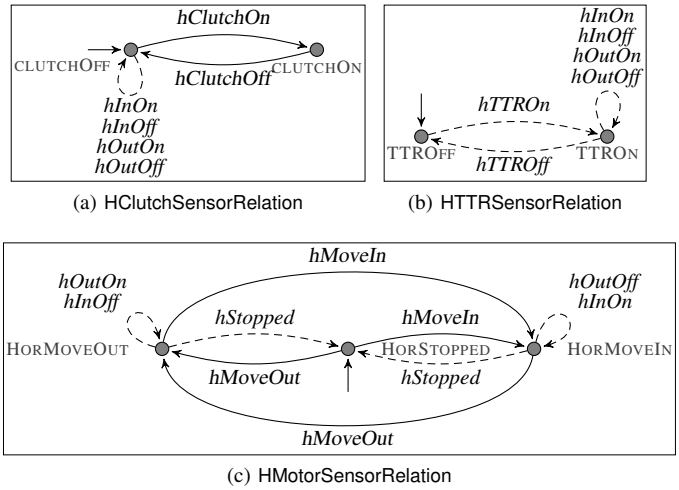
c) Sensors relations (HSensorsRelation): The two end-sensors cannot be active at the same time, as result of their physical location, see Figure 10. Furthermore, the tabletop can only be added and removed in the maximally out position.

d) Sensor-actuator relation (HActuatorSensorRelations): Only when the tabletop is present and is moving horizontally, can the maximally in and out sensors change state. The tabletop can move horizontally in three distinct cases:

- The clutch is released, see Figure 11a; the table can be moved by hand, therefore the sensors can always switch in any order.
- The TTR switch is activated, see Figure 11b; the table can be moved by hand, as in the case that the clutch is released.
- If the clutch is applied, and the TTR switch is not active, the movement is controlled by the motor, see Figure 11c; analogous to the vertical axis.

That is, if either the clutch is released or the TTR switch is activated, then the table can be moved manually, and hence the sensors at the maximally in and maximally out positions can be activated in any order. This behavior can be defined by a single, 12-state, automaton, representing the interleaving (non-synchronising) parallel composition (cartesian product) of the three automata of Figure 10:

HActuatorSensorRelations $\triangleq$ HClutchSensorRelation ||| HTTRSensorRelation ||| HMotorSensorRelation.



(a) HClutchSensorRelation (b) HTTRSensorRelation

(c) HMotorSensorRelation

Fig. 11. Model HActuatorSensorRelations: Actuator sensor relation (interleaving parallel composition)

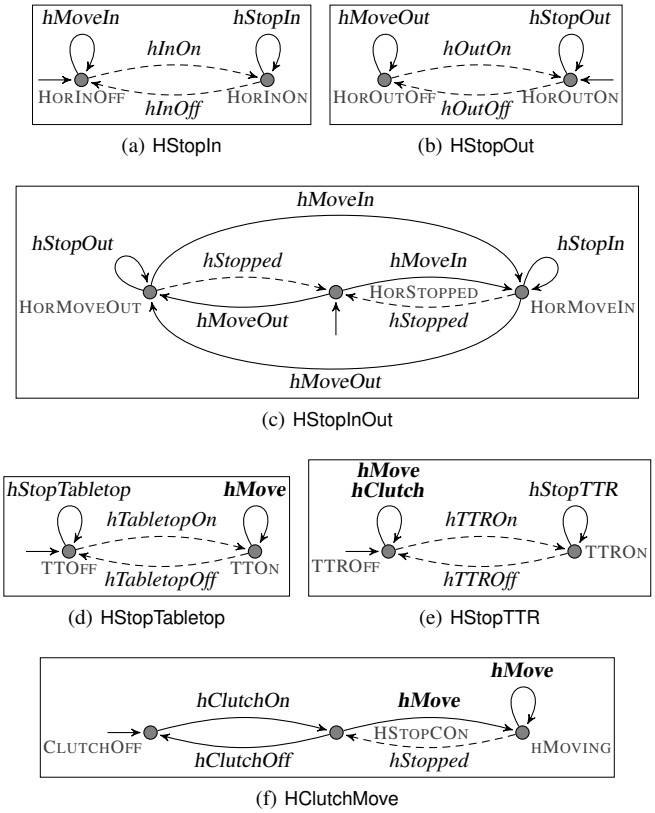
The only shared events in the three automata are the events *hInOn*, *hInOff*, *hOutOn*, *hOutOff*, occurring as self loops. Because the other events (*hClutchOn*, *hClutchOff*, *hTTROn*, *hTTROff*, *hMoveIn*, *hMoveOut*, *hStopped*) each occur in one automaton only, the only difference between the synchronizing and interleaving parallel composition of the three automata is in the self loops: the states, and the transitions between the states are identical. Therefore, the interleaving parallel composition of the three automata is still a deterministic automaton. Each state of the parallel composition (interleaving or synchronous) consists of the three sub-states of the respective automata. The difference between the synchronous and interleaving parallel composition is that the set of self loops of each state of the interleaving parallel composition is obtained by taking the *union* of the sets of self loops of the three sub-states, whereas for the synchronous parallel composition, the *intersection* of the sets of self loops of the sub-states is taken.

Note that if we had used synchronous parallel composition instead of the interleaving one, we would have obtained models which are inconsistent with the physical system. For example, consider the synchronous parallel composition of HClutchSensorRelation and HMotorSensorRelation. Notice that in HClutchSensorRelation, the events *hInOn*, *hInOff*, *hOutOn*, *hOutOff* can occur only in the initial state. In contrast, none of these events can occur in the initial state of HMotorSensorRelation. Hence, none of the events *hInOn*, *hInOff*, *hOutOn*, *hOutOff* can occur in the initial state of the synchronous composition of HClutchSensorRelation and HMotorSensorRelation. This contradicts the physical behavior we want to model.

2) Control requirements (HReq): The requirement model of the horizontal axis consists of the synchronous parallel composition of multiple control requirements, namely:

$$\text{HReq} \triangleq \text{HStopIn} \parallel \text{HStopOut} \parallel \text{HStopInOut} \parallel \text{HStopTabletop} \parallel \text{HStopTTR} \parallel \text{HClutchMove}$$

a) Maximally in and out (HStopIn, HStopOut, HStopInOut): The horizontal axis may not move beyond its



(a) HStopIn (b) HStopOut

(c) HStopInOut

(d) HStopTabletop (e) HStopTTR

(f) HClutchMove

Fig. 12. Model HReq: horizontal control requirements

maximally in and out positions, see Figures 12a–12c.

b) Tabletop move (HStopTabletop): When the tabletop is not present, initiating horizontal movement is not allowed and the motor should be stopped, see Figure 12d. Note that the SCT framework does not allow to explicitly require that an event (stopping of the motor) *must* occur. We modeled this requirement by allowing only this event (HStopTabletop), to occur in the corresponding state. Together with the non-blockingness property of the supervisor and the particular method used to implement the supervisor, this guarantees that the event HStopTabletopwill be executed by the closed-loop system.

c) TTR move and clutch (HStopTTR): Commands for horizontal movement and clutch commands may only be issued when the TTR switch is off, see Figure 12e. However, it cannot be prevented that the TTR switch is turned on while moving. Whenever the TTR switch is turned on, the table should be stopped (*hStopTTR*).

d) Clutch move (HClutchMove): The tabletop may only be moved by the horizontal motor if the clutch is applied, see Figure 12f. If the clutch is not applied, the motor may not move the table. While the motor is moving the table, the clutch may not be released.

C. Horizontal and vertical axis interaction

There is no physical interaction between the transducers of the horizontal and vertical axis. However, the tabletop might collide with the magnet, when moving inward if the table is not maximally up, or the table might be damaged, when moving downward if it is not maximally out. These situations must be prevented. Therefore, either the maximally out sensor or the maximally up sensor must be on, unless the TTR switch is on. If the TTR switch is on, the table can be moved freely by the operator. In this case, the control system cannot prevent the situation in which both sensors are off.

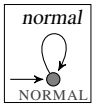


Fig. 13. Model HVNormal: internal event *normal*

1) Plant model (HVNormal): In Figure 13 the internal event *normal* is introduced. This event is used to distinguish two internal states in the requirements, namely, 1) control after TTR is activated, and 2) control after the normal event has occurred, see Figure 14a.

2) Control requirements (HVReq): The control requirements for horizontal and vertical interaction are defined by the synchronizing parallel composition of two models:

$$\mathsf{HVReq} \triangleq \mathsf{HVMode} \parallel \mathsf{HVSafe}$$

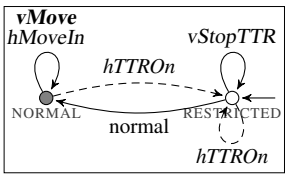
a) Movement restrictions (HVMode): Initially the system is in the state RESTRICTED, see Figure 14a. In this state, the table may not move horizontally inwards (*hMoveIn*), and all vertical movements (***vMove***) are disabled. After the event *normal*, the system enters the state NORMAL, in which all movement events are allowed. After occurrence of the event *hTTROn*, the system enters the state RESTRICTED again.

b) Normal operation (HVSafe): The system can switch to normal operation if it can be ensured that the system stays either maximally out, or maximally up, see Figure 14b. Normal mode is represented by the states V̂HN, VHN and VĤN. In these states it is ensured that the table remains either maximally out or maximally up. The letters V, H and N represent the states vertically maximally up, horizontally maximally out, and normal, respectively. The hat represents negation, e.g. V̂ represents not vertically maximally up. After an event *hTTROn*, any horizontal or vertical position can be reached (corresponding to the states V̂HN̂, VHN̂, VĤN̂ and V̂ĤN̂).

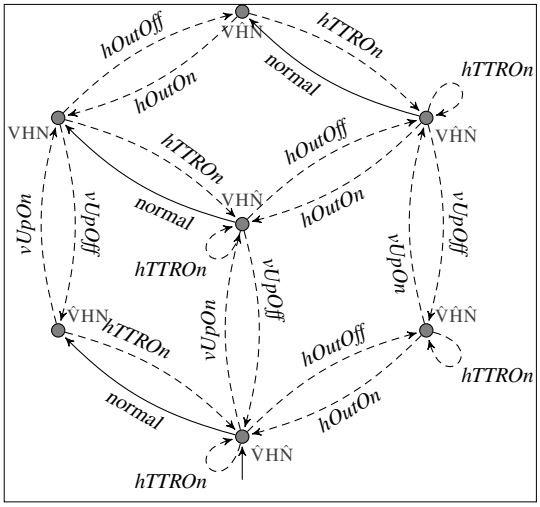
Notice that in Figure 14b state V̂ĤN is not present. The control requirement forbids that this state be reached. Therefore, all events leading to this state are disabled in the requirement model. The controller synthesis algorithm ensures that this requirement is met by disabling only controllable events. For instance, in normal mode when the table is not maximally up, the supervisor will ensure that the clutch is enabled and horizontal movement is prohibited.

D. User interface

The user can control the system by means of a button and a tumble-switch. When the button is pushed, the clutch is released (applied) to switch the table to manual mode (motorized mode). Therefore, this button is called the “manual



(a) HVMode



(b) HVSafe

Fig. 14. Model HVReq: horizontal and vertical restrictions

button". In the motorized mode, the position of the tumble-switch determines the movement of the table. A LED is used to indicate whether the system is in the manual mode or in the motorized operation mode. Note that in manual mode, the supervisor can still prevent the table from performing operations requested by the user, such as moving the table motorized.

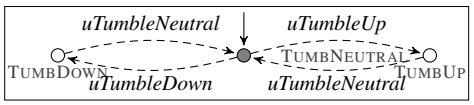
1) Plant model (UI): The user interface only contains external events. The button and switch generate uncontrollable external events. The plant model of the user interface model consists of the synchronous parallel compositions of the actuators:

$$\mathsf{UI} \triangleq \mathsf{UITumbleSwitch} \parallel \mathsf{UIManualButton} \parallel \mathsf{UIManualLED}$$

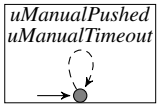
a) Tumble-switch (UITumbleSwitch): The tumble-switch can either be in the position up, down, or neutral, see Figure 15a. When released, the switch returns to the neutral state, as a result of its physical construction. Therefore only state NEUTRAL is marked.

b) Manual button (UIManualButton): When the manual button is pressed, the event *uManualPushed* is emitted and a timer is set. When the timer has elapsed, a timeout event is emitted. However, when the manual button is pressed before the timer has elapsed, the event *uManualPushed* is emitted again, and the timer is set again. This behavior is simplified to one state where the two events are self-looped, see Figure 15b. This simplification turned out to be accurate enough for our purposes.

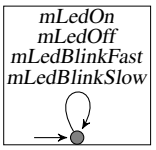
c) LED (UIManualLED): The LED indicates manual or motorized operation mode of the system. It can either be on, off, blinking slowly, or blinking fast. The LED is controlled by events named accordingly, see Figure 15c.



(a) UITumbleSwitch



(b) UIManualButton



(c) UIManualLED

Fig. 15. Model UI: user interface

2) Control requirements (UIReq): The requirement model for the user interface consists of the synchronous parallel composition of multiple control requirements, namely:

$$\mathsf{UIReq} \triangleq \mathsf{UIReqTumble} \parallel \mathsf{UIManualClutch} \parallel \mathsf{UIReqLED}.$$

a) Tumble move (UITumbleMove): The position of the tumble-switch determines which kind of movement of the table is allowed. When the tumble-switch is up, the table is only allowed to move up or to move horizontally into the bore. When the switch is in the down position, the table is only allowed to move down or to move horizontally out of the bore. When the tumble-switch is in its neutral position, all movement should be stopped, see Figure 16a.

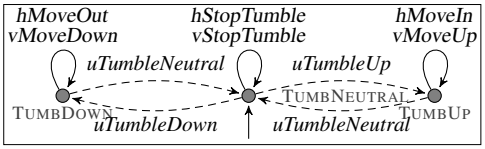
b) Tumble HV switch (UIHVSwitch): If the table is moving up and reaches the uppermost position, the tumble-switch must return to the neutral position before movement into the bore may begin. Similar behavior is required when moving in the opposite direction. These requirements are modeled in Figure 16b.

c) Manual clutch(UIManualClutch): Pushing the manual button results in the *uManualPushed* event, and starts a timer. As a result, the event *hClutchOn* or *hClutchOff* should be triggered, if one of these events is allowed by the other requirements. If none of *hClutchOn* and *hClutchOff* are allowed before the timer expires, the *uManualTimeout* event will be executed instead, see Figure 17.

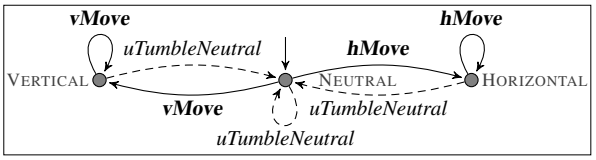
d) LED (UILedModes, UILedClutch): The LED indicates which operating mode is active, and whether the clutch is applied. The LED blinks if the system is in restricted mode, otherwise the LED is on or off, see Figure 18a. If the clutch is applied, the LED is off or blinks slow, see Figure 18b. If the clutch is released, the LED is on or it blinks fast. The operating modes are defined in Figure 18c.

V. EFFECT OF SCT ON EVOLVABILITY

The goal of this section is to present some experimental results on the effect of using SCT on evolvability of the controller. The experiment involved generating a new controller in order to meet a user request for improved functionality. Note that the results reported below cannot be viewed as a systematic evaluation of evolvability; they are intended only as an illustration. In particular, we do not claim that the specific flavour of SCT used for this case study supports evolvability



(a) UITumbleMove



(b) UIHVSwitch

Fig. 16. Model UIReqTumble: Requirements for tumble and clutch operation

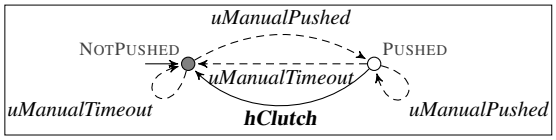
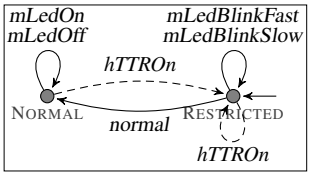


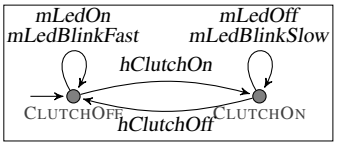
Fig. 17. Model UIManualClutch: Requirements for manual clutch operation

better than other (SCT or non-SCT based) methods. We merely state our experience for the current case study. Note that for the actual experiment, models were used which are slightly different from, but functionally equivalent to, those presented in this paper.

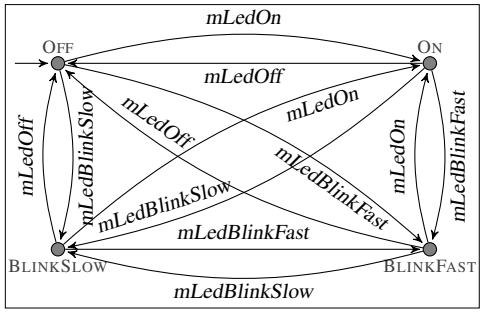
The control requirements presented in Section IV result in a controller that does not allow inward movement of the tabletop when the table is not maximally up. Therefore, if the user switches the tumble-switch up (which corresponds to inward or upward movement of the tabletop) in this state, the tabletop will not move. This behavior of the closed-loop system was considered to be counter-intuitive for the user. The desired new behavior in this case was, that when the tumble-switch was up, the table should move out until it reached the maximally out position, after which the controller should



(a) UILedModes

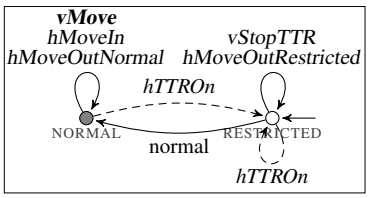


(b) UILedClutch

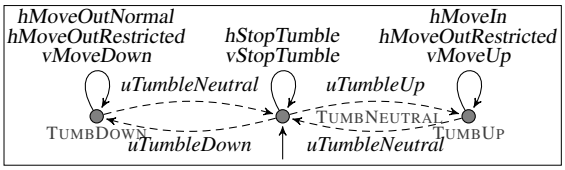


(c) UILedSequence

Fig. 18. Model UIReqLED: Requirements for LED indicators



(a) HVModeNew



(b) UITumbleMoveNew

Fig. 19. Updated requirements for evolvability case

continue its normal operation by moving to the maximally up position, and subsequently moving into the bore. These new requirements were implemented on the actual patient support table in half a day. Implementing the same requirement using the currently used design approach was estimated to take a week. The updated models are presented below.

A. Updated control requirements for the evolvability case

To model the changed requirements, the event *hMoveOut* is defined as an abbreviation of two events in all figures, apart from Figures 14a and 16a: $hMoveOut \triangleq \{hMoveOutNormal, hMoveOutRestricted\}$. Figure 14a is adapted so that the event *hMoveOutNormal* can occur only in mode Normal, whereas the event *hMoveOutRestricted* can occur only in mode Restricted, leading to Figure 19a. In Figure 16a, the event *hMoveOut* is replaced by *hMoveOutNormal*, and the event *hMoveOutRestricted* is added to the self-loops in states TumbleDown and TumbleUp, leading to Figure 19b. In this way, operating the tumble switch in either direction in Restricted mode always leads to the only allowed safe movement: out of the bore.

VI. Generation, implementation and validation of the control software

The plant models and control requirements were modeled using the SCIDE (Supervisory Control Integrated Development Environment) tool set. Subsequently, in SCIDE, a corresponding supervisor was generated using the SuSyNA (Supervisory control Synthesis of Nondeterministic Automata) tool set. It contains an optimized implementation of the algorithms described in [23]. Calculation took a few seconds on a Core 2 Duo, 3Ghz, 3Gb computer. The resulting supervisor contains 30,880 states and 264,456 transitions. The SCIDE and SuSyNA tools have been superseded by the tools for the Compositional Interchange Format (CIF, see [24], [25]).

A. Validation of functional correctness

Although supervisory control theory ensures that the controller satisfies the control requirements by construction, it remains a non-trivial task (but still easier than the development

of the supervisory controller itself) to define the correct plant and requirement models. Thus, errors or undesired behavior may still be present in the plant models and/or requirement models. To help validate the controlled system, we use the framework of [26]. This framework is based on the model-based engineering paradigm, where models are the primary artifacts in the design process. Below we give a brief overview of the framework, for details see [26]. We would like to remark that the framework is a general one and has already been applied to other case-studies.

The design process of [26] consists of the following steps:

A) Modeling of the uncontrolled plant and control requirements.
B) Synthesis of the supervisory controller using the models from step A, resulting in a model of the supervisor.
C) Simulation (un-timed) of the parallel composition of the plant models from step A, and the supervisor obtained in step B.
D) More detailed, e.g. timed or hybrid, modeling of the plant models.
E) Simulation (timed or hybrid) of the model of the closed-loop system which is obtained by combining the plant model from step D with the supervisor from step B.
F) Real-time simulation involving the actual plant hardware (the uncontrolled system) coupled with the model of the supervisor obtained in step B.
G) Code generation from the supervisor model obtained in step B.
H) Real-time control of the plant hardware controlled by the realization of the supervisor obtained by step G.

The framework consists of:

- Modeling environments to support the design steps A and D.
- Transformation tools to transform:
  1) The models of step A to input models of the synthesis tools used in step B. This transformation amounts to transforming the model represented in one modelling formalism to an equivalent model in another modelling formalism. This transformation is a purely syntactic one.
  2) The models of steps A, B, and D to input models of the simulation tools used in steps C and E.
- An infrastructure to couple models and realizations of components for step F.
- Code generators to generate code (step G) from the supervisor model obtained in step B.

The transformation and simulation tools that are described in the preceding steps are all based on the CIF, see [24], [25]. Steps C, E, and H are described in more detail in the sections below.

B. Untimed simulation

Although the supervisor obtained using the SCT framework is guaranteed to meet the formalized control requirements, this does not yet mean that the plant will meet the initial (informal) control requirements. This can happen due to inadequate modelling of the plant or control requirements. Hence, it is still necessary to validate the correctness of the synthesized controller. In case of an incorrect controller, one immediately knows that the mistake must be in the model of the uncontrolled plant or in the control requirements.

To validate the synthesized controller, the state space of the model of the *controlled system* is explored by hand, using user guided simulation. That is, based on different scenarios of occurrence of events, events are chosen manually and executed. By scenarios we mean sequences of external events, which the environment is expected to generate. The scenarios are determined manually, based on the domain knowledge. We would like to stress that scenarios are used to *test* the correctness of the controller, not to *design* the controller.

C. Hybrid simulation

To validate the dynamic behavior of the plant controlled by the synthesized supervisor, the CIF model of the supervisor is simulated together with (using synchronizing parallel composition) a more detailed, hybrid model of the plant. The latter model is developed separately from the discrete-event model used for generating a supervisor, and incorporates more details, for example, the timing aspects. During simulation, the reaction of the model to various sequences of *external events* (i.e. events generated by the environment, such as error conditions, operator actions, etc.) is evaluated. The models of the plant and the various external events are specified in CIF, modeling both discrete-event and continuous-time behavior.

In order to facilitate timed simulation of the closed-loop system, the timing of the events generated by the supervisor should be presented. Note that in SCT framework, the supervisor is assumed to be untimed and in fact it is assumed that it enables/disables events, rather than generating them. Hence SCT does not tell us how to interpret the supervisor's behavior in a timed setting.

We interpret supervisors in a timed environment as follows. The events present in the discrete-event plant model are taken to be *urgent*, see [27], in the hybrid model. That is, first all the events which are enabled by the supervisor and which can be executed immediately from the current state of the plant are executed. During the execution of these events, time is not allowed to progress. When there are no more events which are both enabled and can be executed immediately, then time is allowed to progress again.

The CIF simulator can operate in interactive or automatic mode, which differ in how they select the event to be executed, in case more than one event is enabled. In interactive mode, the user selects the event to be executed, whereas in automatic mode, the simulator selects the event to be executed: there are options to select the first event, the last event, or to perform a random selection.

The simulation step described above can be used to check if the closed-loop system is safe. However, simulation does not entirely guarantee correctness, as the interaction between the hybrid model of the plant and the supervisor differs from the interaction between the implementation of the supervisor and the physical plant. In particular, where in the actual implementation, enabled controllable events are executed when there are

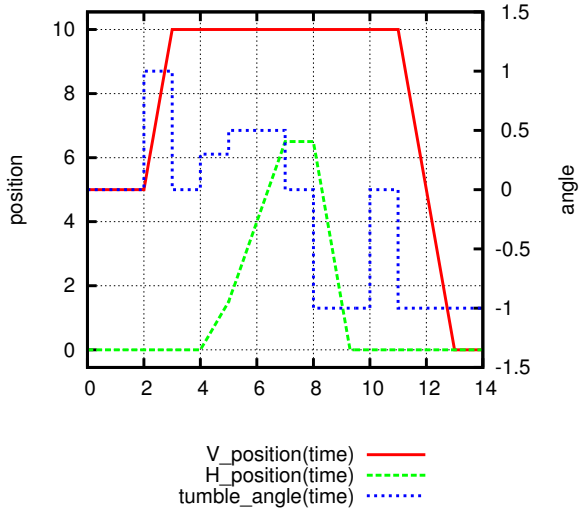


Fig. 20. Simulation results tabletop movement controlled by tumble-switch.

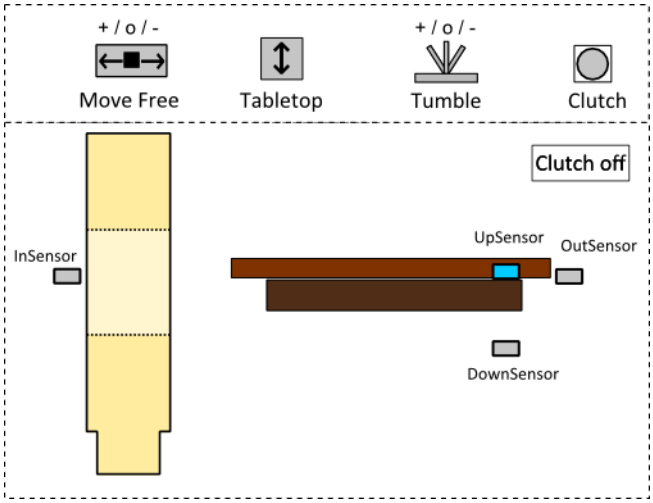


Fig. 21. SVG image for real-time, interactive, simulation and animation.

no more enabled uncontrollable events, in the simulation this need not be the case.

Figure 20 shows the simulation results as a function of time of a representative case where the horizontal and vertical movement of the table-top is controlled by the tumble-switch. The position of the tumble-switch ranges from -1 to + 1. When the tumble-switch is released, its position is 0. Initially, the table is halfway up, and the tabletop is placed onto the table at the maximally out position. The tumble-switch is used to move the table to the upper position. When the table reaches the upper position at time 3, the table stops, and the tumble-switch is released. Then the table is moved inward, first slowly, then faster. After that, at time 7, the movement is stopped. Then, the table is moved out , until the table reaches the maximally out position, where it stops at time 9. The tumble-switch is momentarily released (for 1 time-unit), causing downward table movement until the table reaches its lowest position.

Apart from output of the model variables in a graph as a function of time, as shown in Figure 20, the CIF simulator also supports real-time, interactive, simulation and animation, based on user supplied images of the system in the standardized SVG (Scalable Vector Graphics) format [28], as shown in Figure 21. To ensure that the simulation model can be used in its original form, without user defined animation statements, we need a so called CIF/SVG Mappings file [25]. This user-supplied text file contains commands that define connections between the dynamic state of the simulation model (e.g. the value of a differential variable that models the position of the table top) and attributes of objects (e.g. the graphical position of the table top) in the user supplied SVG drawing. User interaction with the simulation by means of, for example, pressing buttons in the animation, is interfaced with the simulation by means of execution of events named in the CIF/SVG Mappings file.

D. Real-time implementation

When implementing supervisors for actual real-time control, three issues need to be dealt with:

1) The SCT framework is untimed. Implementations, however, are timed. This issue was first discussed in detail by [29].
2) Supervisory control theory is based on synchronous execution of events shared between supervisor and plant. This leads to the research question how to implement a synchronous system in an asynchronous way. Recent work on desynchronisation techniques is available in [30], [31].
3) Synthesis generates a set of maximally permissive supervisors, so that for an implementation, choices need to be made. Furthermore, in the original supervisory control theory, events originate from the plant, and can be disabled or enabled by the supervisor. Real-time controllers, however, actually generate events. To support synthesis of controllers, as opposed to supervisors, [32] proposes forced events, [33] proposes a Ξ subset of controllable events that are initiated by the controller, and [34] proposes directed controllers, that select at most one controllable event to be enabled at any instant. Another option is to model the plant as a transducer and then translate the problem to a classical SCT problem, see [35]. In fact, the framework of [35] was applied to small academic example inspired by the current case study.

An overview of several of the issues discussed above, in the context of PLC-based supervisory controller implementations, is given by [36].

Despite the existence of the literature cited above, we did not use it for implementing the controller. The reason for this was purely pragmatic: we had access to the hardware for a limited period of time, and this period was not sufficiently long for exploring the methods described in the literature. Moreover, the ad-hoc solution to be described below appeared adequate for our purposes.

In the actual real-time implementation of the patient support system, the sensors and actuators are connected to an industrial grade control unit. This control unit is connected to a standard PC by means of an IEEE 1394 FireWire high speed serial

bus interface. The control unit conditions the sensor signals, and takes care of motion and I/O control. Furthermore, it translates sensor state changes to (uncontrollable) events and it executes the high-level commands (controllable events) generated by the PC. On the PC, the events from the control unit are buffered in an input event queue, and handled by an event handler. After executing an event from the input event queue, the state of the supervisor is updated. If the event queue is empty, the set of controllable events that is allowed by the supervisor in the current state is calculated. From this set, an event is selected and sent to the control unit for execution. In this way, nondeterminism is resolved by giving priority to uncontrollable events over controllable events. This strategy is based on the assumption that the uncontrollable events are properly conditioned by the control unit, ensuring a minimum time interval between successive uncontrollable events from each component. Switches would need to be properly debounced, and switching the tumble switch from the up position, via the neutral position to the down position, for instance, would mean generation of a sequence of the *uTumbleNeutral* and *uTumbleDown* events separated by a minimum time interval, giving the supervisor time to empty the input queue after receipt of the *uTumbleNeutral* event, and to subsequently execute the *hStopTumble* or *vStopTumble* event. Considering various other strategies for resolving nondeterminism is the subject of future research. As has been pointed out in [37], the need for the implementation of the supervisor to pick an enabled event according to some mechanism, may cause the closed-loop system to become blocking. For our particular case-study, the closed-loop system remained non-blocking. Finding a theoretical explanation for this remains a topic of further research.

All controllable events are initiated by the supervisor, that runs on the PC, and all uncontrollable events are generated by the control unit, that is connected to the patient support table. All events are defined as urgent, so that they are executed as soon as they are enabled.

VII. Concluding remarks

We have reported on a successful industrial application of supervisory control theory (SCT) to synthesize a supervisory controller for real-time control of a patient support system of an MRI scanner. The use of SCT and tools enables easy adaptation to changing control requirements. In the case of such a change, only the new requirements need to be formally defined. After the formalization step is completed, the theory and tools provided by the supervisory control framework allow automatic generation of suitable control software. Easy adaptation of the specifications was further aided by a modular approach to the definition of plant models and control requirements.

Acknowledgment

The authors thank the anonymous referees for the detailed comments that have been of great help in improving the drafts of this article. Furthermore, they thank Albert Hofkamp and Dennis Hendriks for their contribution to the development of the software tools, and Rong Su for his help with the theoretical aspects of SCT.

References

[1] W. M. Wonham, *Supervisory control of discrete-event systems*. Toronto, ON, Canada: Dept. Elect. Comput. Eng., Univ. Toronto, 2012. [Online]. Available: http://www.control.toronto.edu/DES/

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York: Springer, 2007.

[3] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, 1993.

[4] J. Liu and H. Darabi, "Ramadge-Wonham supervisory control of mobile robots: lessons from practice," in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 1, Washington, 2002, pp. 670–675.

[5] K. T. Seow and M. Pasquier, "Supervising passenger land-transport systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 3, pp. 165–176, 2004.

[6] M. Moniruzzaman and P. Gohari, "Implementing supervisory control maps with PLC," in *Proceedings American Control Conference*, New York, 2007, pp. 3594–3599.

[7] M. Noorbakhsh and A. Afzalian, "Design and PLC based implementation of supervisory control for under-load tap-changing transformers," in *Proceedings International Conference on Control, Automation and Systems*, Seoul, 2007, pp. 901–906.

[8] R. J. Leduc and W. M. Wonham, "Discrete event systems modeling and control of a manufacturing testbed," in *Canadian Conference on Electrical and Computer Engineering*, vol. 2, Montreal, 1995, pp. 793–796 vol.2.

[9] B. A. Brandin, "The real-time supervisory control of an experimental manufacturing cell," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 1–14, 1996.

[10] S. C. Lauzon, A. K. L. Ma, J. K. Mills, and B. Benhabib, "Application of discrete-event-system theory to flexible manufacturing," *IEEE Control Systems Magazine*, vol. 16, no. 1, pp. 41–48, 1996.

[11] A. Hellgren, M. Fabian, and B. Lennartson, "Modular implementation of discrete event systems as sequential function charts applied to an assembly cell," in *Proceedings of the IEEE International Conference on Control Applications*, Mexico City, 2001, pp. 453–458.

[12] S. Kim, J. Park, and R. C. Leachman, "A supervisory control approach for execution control of an FMC," *International Journal of Flexible Manufacturing Systems*, vol. 13, no. 1, pp. 5–31, 2001.

[13] V. Chandra, Z. Huang, and R. Kumar, "Automated control synthesis for an assembly line using discrete event system control theory," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 33, no. 2, pp. 284–289, 2003.

[14] M. Nourelfath and E. Niel, "Modular supervisory control of an experimental automated manufacturing system," *Control Engineering Practice*, vol. 12, no. 2, pp. 205–216, 2004.

[15] O. Ljungkrantz, K. Åkesson, J. Richardsson, and K. Andersson, "Implementing a control system framework for automatic generation of manufacturing cell controllers," in *2007 IEEE International Conference on Robotics and Automation*, Roma, 2007, pp. 674–679.

[16] J. Pétin, D. Gouyon, and G. Morel, "Supervisory synthesis for product-driven automation and its application to a flexible assembly cell," *Control Engineering Practice*, vol. 15, no. 5, pp. 595–614, 2007.

[17] Í. Hasdemir, S. Kurtulan, and L. Gören, "An implementation methodology for supervisory control theory," *The International Journal of Advanced Manufacturing Technology*, vol. 36, no. 3, pp. 373–385, 2008.

[18] D. B. Silva, A. D. Vieira, and E. F. R. Loures, "Dealing with routing in an automated manufacturing cell: a supervisory control theory application," *International Journal of Production Research*, vol. 49, no. 16, pp. 4979–4998, 2011.

[19] R. J. M. Theunissen, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda, "Supervisory control synthesis for a patient support system," in *Proceedings of the European control conference*, Budapest, Hungary, 2009, pp. 4647–4652.

[20] R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda, "Application of supervisory control synthesis to mri scanners: improving evolvability," Eindhoven University of Technology, The Netherlands, SE Report 2010-06, 2010. [Online]. Available: http://se.wtb.tue.nl/sereports

[21] R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda, "Improving evolvability of a patient communication control system using state-based supervisory control synthesis," *Advanced Engineering Informatics*, vol. 26, no. 3, pp. 502–515, 2012.

[22] D. A. van Beek, P. Collins, D. E. Nadales Agut, J. E. Rooda, and R. R. H. Schiffelers, "New concepts in the abstract format of the Compositional Interchange Format," in *3rd IFAC Conference on Analysis and Design of Hybrid Systems*, A. Giua, C. Mahuela, M. Silva, and J. Zaytoon, Eds., Zaragoza, 2009, pp. 250–255.

[23] R. Su, J. H. van Schuppen, and J. E. Rooda, "Aggregative synthesis of distributed supervisors based on automaton abstraction," *IEEE Transactions on Automatic Control*, vol. 55, no. 7, pp. 1627–1640, 2010.

[24] D. E. Nadales Agut, D. A. van Beek, and J. E. Rooda, "Syntax and semantics of the compositional interchange format for hybrid systems," *Journal of Logic and Algebraic Programming*, vol. 82, no. 1, pp. 1–52, 2013.

[25] Systems Engineering Group TU/e, "CIF toolset," http://cif.se.wtb.tue.nl, 2013.

[26] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda, "Model-based engineering of supervisory controllers using CIF," in *Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling*, ser. Electronic Communications of the EASST, vol. 21, Denver, 2009, pp. 1–10.

[27] D. A. van Beek, P. J. L. Cuijpers, J. Markovski, D. E. Nadales Agut, and J. E. Rooda, "Reconciling urgency and variable abstraction in a hybrid compositional setting," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS, K. Chatterjee and T. Henzinger, Eds. Springer, 2010, vol. 6246, pp. 47–61.

[28] W3C, "W3C SVG working group," http://www.w3.org/Graphics/SVG/, 2011.

[29] S. Balemi, "Input/output discrete event processes and communication delays," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 4, no. 1, pp. 41–85, 1994.

[30] H. Beohar and P. J. L. Cuijpers, "Desynchronizability of (partial) synchronous closed loop systems," *Scientific Annals of Computer Science*, vol. 21, no. 1, pp. 5–38, 2011.

[31] H. Beohar, "Refinement of communication and states in models of embedded systems," Ph.D. dissertation, Eindhoven University of Technology, 2012.

[32] C. H. Golaszewski and P. J. Ramadge, "Control of discrete event processes with forced events," in *Proceedings 26th IEEE Conference on Decision and Control*, vol. 26, Los Angeles, 1987, pp. 247 – 251.

[33] P. Dietrich, R. Malik, W. Wonham, and B. Brandin, "Implementation considerations in supervisory control," in *Synthesis and Control of Discrete Event Systems*. Kluwer, 2002, pp. 185–201.

[34] J. Huang and R. Kumar, "Optimal nonblocking directed control of discrete event systems," *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 4, pp. 620 – 629, 2008.

[35] M. Petreczky, R. J. M. Theunissen, D. A. van Beek, R. Su, J. H. van Schuppen, and J. E. Rooda, "Control of input-output discrete-event systems," in *Proceedings European Control Conference*, Budapest, 2009, pp. 1–6.

[36] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proceedings 37th IEEE Conference on Decision and Control*, Tampa, 1998, pp. 3305–3310.

[37] P. Malik and R. Malik, "Modular control-loop detection," in *8th International Workshop on Discrete Event Systems*, 2006, pp. 119–124.

Rolf Theunissen Rolf Theunissen received the MSc degree from Eindhoven University of Technology. Currently he is finishing his PhD project at the Eindhoven University of Technology, on the topic of supervisory control. In 2011, he joined Nspyre where he works on the development of DSLs, model transformations and analysis tools, currently focused at an integrated development flow for servo controllers of litho scanners.

Mihaly Petreczky Mihaly Petreczky received a M.Sc. in computer science and a PhD in mathematics from Vrije Universiteit in Amsterdam, The Netherlands in 2002 and in 2006 respectively. In the past, he held appointments as a postdoc at Johns Hopkins University, USA, Eindhoven University of Technology, The Netherlands, and as an assistant professor at Maastricht University, The Netherlands. He is currently an assistant professor at Ecole des Mines de Douai, France. His research interests include control and systems theory of hybrid and discrete-event systems.

Ramon Schiffelers Ramon Schiffelers received the MSc and PhD degrees from Eindhoven University of Technology. His PhD project resulted in the hybrid process algebra Chi and accompanying tools. As a post-doctoral researcher, he dealt with modeling, analysis and synthesis of supervisory controllers for Philips MRI scanners and participated in European projects dealing with the design of the Compositional Interchange Format for hybrid systems. In 2010, he joined ASML where he is responsible for the development of DSLs and analysis tools to enable an integrated development flow for servo controllers of litho scanners. Since 2012 he also holds a position as assistant professor in de Model Driven Software Engineering group at the Department of Mathematics and Computer Science at the Eindhoven University of Technology.

Bert van Beek Bert van Beek received the M.Sc. degree in electrical engineering in 1985 and the PhD degree in 1993, both at the Eindhoven University of Technology, where he has been employed as an assistant professor at the Department of Mechanical Engineering since 1986. He is one of the founders of the Compositional Interchange Format, and has extensive experience in supervisory control for industrial applications. His research interests include modeling, simulation, controller synthesis and verification of hybrid industrial systems, and in particular of embedded systems. He has been involved (and is currently involved) in several European projects, including HYCON, Twins, C4C, Multiform, and HYCON2.

Koos Rooda Jacobus (Koos) E. Rooda is emeritus professor Systems Engineering. In 1971 he received his MSc degree in Food Technology from Wageningen University of Agriculture, The Netherlands. From 1971 until 1985 he was assistant professor and was active in the area of Internal Transport Systems of the Mechanical Engineering Department at Twente University, Enschede, The Netherlands. In 1978 he received his PhD degree also at Twente University. In 1985 he became full professor in Systems Engineering in the Department of Mechanical Engineering of Eindhoven University of Technology. He is a member of IEEE, ACM, IFAC, IIE and SCS. He retired on April 1, 2010. His interests include analysis of discrete-event and continuous-time systems with a concurrent behavior, as well as supervisory control of (manufacturing) machines and control of (manufacturing) networks.