Application of supervisory control synthesis to MRI scannes : improving evolvability

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2010-06

Application of supervisory control synthesis to MRI scanners: improving evolvability

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

ISSN: 1872-1567

SE Report: Nr. 2010-06
Eindhoven, April 2010
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

This paper presents an application of Ramadge-Wonham supervisory control theory to a patient support system for MRI scanners. The obtained controller was implemented and tested on real hardware. A distinguishing feature of the proposed application is that the evolvability of the controller is an essential requirement. By evolvability is the ability to adapt the controller to meet modified control objectives. The experiments conducted in the course of the case-study indicate that the use of supervisory control theory improves the evolvability of the controller.

1 Introduction

The goal of this paper is to present a specific application of Ramadge-Wonham supervisory control theory (SCT), see [31, 8], to control problems arising in high-tech systems. The application addresses the control of a patient support table for MRI scanners of Philips Medical Systems. More precisely, the task was to design a controller for this patient support table such that the following requirements are satisfied.

- **Functional correctness**
  The closed-loop system satisfies the specified control objectives
- **Evolvability**
  Roughly speaking, evolvability means the following. If the control objectives change, then the controller can be easily adapted to meet the new control objectives. The concept of evolvability is discussed in more detail later on.

The control objectives are of a discrete nature and describe the criteria of functional correctness of the system. Informally, the control objectives require that the system stays safe and that it does not enter a deadlock state. Notice that the controller is not only required to ensure correct functionality. It should also be easily adaptable to changing control requirements. I.e. it should be *evolvable*. This emphasis on evolvability renders the considered control problem quite different from other applications of supervisory control theory. In particular, the requirement of evolvability means that the control synthesis method should keep the amount of manual tuning to a minimum and facilitate systematic (re)use of models.

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 was created. Only the behavior of the plant in the absence of errors is modeled. Second, the control requirements were modeled as a 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 on the real hardware. The evolvability of the controller was tested by redesigning it for slightly different control requirements; the design and implementation of the new controller took only *four hours*, see Section 6 for more details.

This paper explains the original control problem and present certain components of the plant model and of the model of requirements. The modeling choices and the procedure and software which was used to obtain a supervisor are discussed. Finally, the implementation and the validation of the supervisor, and the effect of using supervisory control theory on evolvability are described.

In addition, some guidelines are formulated for modeling high-tech systems such that the obtained models fit the theoretical framework of supervisory control. Although the guidelines are not claimed to be applicable in general, the guidelines turned out to be useful for the particular case-study.

It is worth noting that not only a supervisory controller is 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 which was implemented on the real hardware. These modifications were applied to render the models easier to understand. These modifications 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.

General goal: improving evolvability of high-tech systems The contribution of the paper is part of a wider research program, which aims at improving of evolvability of *high-tech* systems. High-tech systems are complex systems which combine mechanical and electrical components with electronics and software. Examples of such systems are MRI scanners, printers and manufacturing lines. The design of high-tech systems is a complex process which requires a multidisciplinary team of engineers. In addition, for several classes of high-tech systems, new generations of such systems have to be designed in short time, and with low costs and high quality, in order to meet the market demands. These systems are typically manufactured in relatively small numbers and new versions of them have to be developed every few years. Each new version retains a great deal of functionality of the previous generation. However, the desired new features require the re-design of the control software of the previous generation. In turn, the design of control software is often one of the most time consuming and costly stages of the product development process. Note that control software refers to the software which controls high-level behavior of the hardware. Controllers of various standard hardware components such as electrical motors are deliberately excluded. The problem described above calls for improvement of evolvability of control software, i.e. the improvement of the ability to quickly adapt existing control algorithms to meet slightly changed control requirements.

Supervisory control to improve evolvability of high-tech systems Supervisory control theory (SCT), see [31], is the branch of control theory dealing with control problems where the plant is discrete-event and the control objectives involve only discrete (logical) requirements. It is proposed to use control theory in general, and SCT in particular, to improve evolvability. 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 control requirements, such that the generated controller is guaranteed to meet the control requirements. The envisaged procedure for quick adaptation of controllers is then as follows.

Step 1. A model of the plant and control requirements is created.

Step 2. If the control requirements change, then their models are adapted or extended accordingly.

Step 3. A controller is generated using SCT (and the corresponding tools) from the modified model of control requirements and from the original (or modified) plant model. SCT guarantees that the thus obtained controller meets the (modified) control requirements.

That is, instead of modifying the existing controller, it is proposed to modify the model of the control requirements and automatically generate a new controller, which is guaranteed to meet the modified control requirements.

Differences with respect to classical control problems Control theory has been actively applied in almost every branch of industry. However, traditionally it was used only to obtain a solution to a specific control problem, evolvability of the solution was not considered as a major part of the task. The application of control theory to improve evolvability leads to challenges which are different from the challenges usually encountered in classical control

engineering. In contrast to classical control, *the challenge is not so much to come up with a control algorithm, but to come up with a method for fast adaptation of the control algorithms to the ever changing control requirements*. This calls for systematic methods for modeling the plant and control requirements, and for generating the desired controller. The latter is provided by the existing tools for supervisory control synthesis. The former remains a challenge. Note however, that the proposed guidelines, while not universal, might help to make the modeling effort more systematic.

Related work Despite the fact that SCT is well established, the number of industrial applications is limited. The main causes of this can be found in the computational complexity, model interpretation and modeling and implementation effort, see [1, 18]. Without claiming for completeness, previous applications of SCT include: a rapid thermal multiprocessor, see [1], mobile robots, see [18, 15], passenger land-transport system, see [26], a water bath boiler, see [20], under-load tap-changing transformers, see [21], automated manufacturing and assembly systems: [6, 17, 7, 16, 13, 14, 11, 9, 24, 22, 10, 23, 12].

To the best of our knowledge, the application domain of MRI scanners is new. In addition, none of the above cited papers consider the problem of evolvability. In contrast, improving evolvability is one of the main tasks in the application considered in this paper.

Some of the results of the paper were announced in the conference proceedings [30]. With respect to [30], the main difference is that this paper presents more detailed models and a modeling methodology, in addition this paper systematically explains the role of supervisory control in improving evolvability. The paper [30] can be viewed as a shorter version of the current paper. Some of the results described in this paper were also included in the technical report [29]. However, [29] presents the models and the solution in much less detail, it does not present a modeling methodology, nor it explains the link between evolvability and supervisory control.

Outline The outline of the paper is as follows. Section 2 presents the modeling approach that is used to model the plant. Section 3 contains an informal description of the case-study. Section 4 presents the formal models of some of the plant components and control requirements. The automatic generation of the supervisory controller, its implementation, and the validation of the obtained controller by means of simulation are all discussed in Section 5. Section 6 evaluates the effect of the use of supervisory control on evolvability. Section 7 presents conclusions and proposes directions for future research.

2 Modeling approach

The goal of this section is to present guidelines for developing the model of the plant and the model of the control requirements. Both the plant and the requirements are modeled as finite-state automata.

2.1 Method for constructing the plant model

In theory, the goal of the plant model is to capture the dynamic behavior of the un-actuated systems, i.e. the system without control. That is, the plant model should contain the answer to the question of what the effect is of the past **actuator events** (i.e. inputs generated by the environment or the controller) on the future **sensor events** (i.e. outputs). This information is

necessary in order to construct a controller (supervisor) which, using control inputs, steers the system in such a way that the outputs satisfy the control requirements. Finally, the **internal structure** of the system describes the effect of the actuators on sensors and on **control modes**. A control mode is an entity which contains the information relevant for formulating the control objectives. In contrast to sensors and actuators, control modes need not represent physical entities. Control modes are introduced by the design engineer to facilitate the formulation of control requirements.

From a modeling perspective, the plant is viewed as a parallel interconnection of several components. The components are divided into the following categories.

Models of actuators As the name suggests, these components model the behavior of actuators.

Models of sensors These components model the behavior of sensors.

Models of the internal structure These components represent the "glue" of the model, relating actions of actuators to activations of sensors and to changes in control modes.

The states of actuator models, sensor models and models of the internal structure are referred to as *actuator states*, *sensor states* and *internal states*, respectively. The overall state of the system is the collection of the local states of its components. Hence, *actuator states, sensors states and internal states are components of the global state of the system*.

Another important aspect of modeling is the choice of events and the division of events into controllable and uncontrollable ones. The events are proposed to be grouped according to the following criteria.

Actuator events The actuator events represent the actions of the actuators. These events can be either *controllable* or *uncontrollable*. The former represent those actuator actions which can be triggered by the controller, i.e. *controllable actuator events represent the control inputs*. In contrast, uncontrollable actuator events correspond to the actions which are imposed by the environment, i.e. *uncontrollable actuator events represent external or user inputs, imposed on the system by the environment or user*. The controller cannot trigger or prevent the occurrence of such events.

The common feature of all actuator events is that in principle, the events can take place at any moment of time, and their occurrence triggers a change in the state of the system. More precisely, actuator events trigger changes of the actuator states and internal states of the system. However, these events do not directly change the sensor state component of the global state.

Sensor events The sensor events represent the activation of certain sensors. These events are always *uncontrollable*. Although the occurrence of these events may coincide with changes in the state of the system, *sensor events do not trigger a state transition, sensor events at most indicate that a state transition has occurred*. The latter is the main intuitive difference between sensor events and uncontrollable actuator events. Sensor events indicate the change of the sensor and internal state component of the global state.

Internal events Internal events are events which are not associated with sensors or actuators. These events typically occur within the component describing the internal structure of the system. Internal events can be virtual, in a sense that the events need not be associated with any specific physical entity. Internal events trigger a change in the internal state component of the global state. Internal events do not trigger any change in the actuator or sensor states of the system, nor are these events triggered by a change in

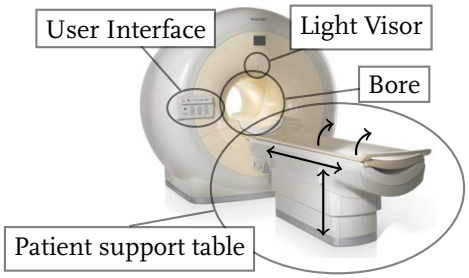


Figure 1: MRI scanner

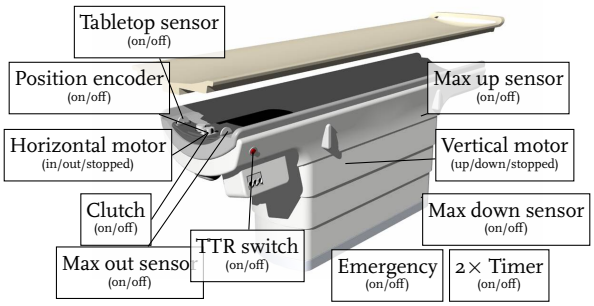


Figure 2: Patient table

the actuator and sensor states. Internal events can be either *controllable* or *uncontrollable*. Controllable internal events could be thought of as internal commands for the potential supervisor (controller), controllable internal events do not change the behavior of the physical plant.

2.2 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.

Notice that the models of control requirements may involve user inputs or external inputs (uncontrollable actuator events), while the physical closed-loop system can actively generate control actions (controllable actuator events), sensor events and internal events, but not external or user inputs. This may look like a contradiction. However, a well-formed model of control requirements should not impose constraint on uncontrollable actuator events, it should use them only to formulate conditions for occurrence of certain sensor or internal events. In fact, supervisory control theory ensures that if this is not the case, i.e. if control requirements impose a constraint on the sequence of these events, then no supervisor exists which solves the corresponding supervisory control problem. In other words, incorrect models of control requirements which impose constraints on uncontrollable actuator events are *automatically detected*.

3 Informal introduction to the case-study

The patient support system is used to position a patient in an MRI scanner, see Figure 1. An MRI scanner is used mainly in medial 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 of a motor drive depending on the state of the clutch. It contains sensors to detect the presence of the tabletop, and the position of the tabletop. Furthermore, the system is equipped with hardware safety systems (emergency stop and tabletop release (TTR)), that allow the operator to override the control system in emergency situations. Finally, the system contains a light-visor for marking that part of the patient's body which is supposed to be scanned. This marking is then used by the MRI scanner to determine the correct position of the patient in the bore. 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. Pushing the light-visor button, should enable the light-visor, and when the button is pushed again, the position is stored for automatic positioning. Finally, the table should not move beyond its end positions, and it should not collide with the magnet.

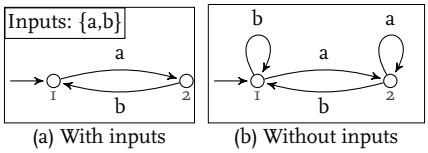
A subset of the functionality of the patient support systems is discussed in this paper. The emergency system and the light-visor with automated positioning, are not modeled. Furthermore, only good weather behavior is considered. Errors in sensors and actuators and disturbances in the system are not modeled.

The patient support system is more difficult to control than it might appear at first sight. It contains several complex interactions of components, and the overall finite state model of the uncontrolled system contains $6.3 \cdot 10^9$ states ($64 \cdot 10^6$ states without user-interface). The complexity of the use case becomes even more apparent when one considers the time which is required to build the control software manually. In fact, it was estimated one would need a week for manual adaptation of the control software to meet the modified requirements described in Section 6. Note that with the approach of this paper, the adaptation of the models of control requirements and the generation of new control software took merely four hours.

4 Formal models

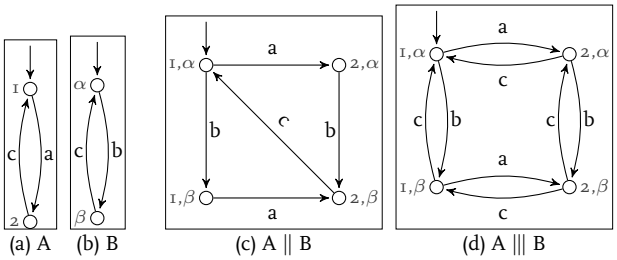
The goal of this section is to describe the formal models of the plant and of the control requirements. In the models, states are denoted by vertices, initial states are indicated by an unconnected incoming arrow, and marked states are denoted by filled vertices. Controllable and uncontrollable events are depicted by solid and dashed edges, respectively. Multiple events on an edge represent an edge for each event. A bold event name indicates a set of events, representing an edge for each event in the set.

For notational purposes, the concept of input events is introduced. Input events are denoted



(a) With inputs (b) Without inputs

Figure 3: Translation of inputs



(a) A (b) B (c) A || B (d) A ||| B

Figure 4: synchronous and interleaving parallel composition

in the upper part of the automaton, indicated by the keyword *Inputs*. The occurrence of the events in this set are not restricted by the automaton model. More precisely, to each state one adds a self-loop, labeled by those input events, for which there exists no outgoing transition from that state labeled by the designated event. Figure 3 shows an example of how input events are translated.

A model may consist of several automata. To compose these automata, two parallel composition operators can be used: 1) the *synchronizing parallel composition operator*, denoted by ||, which requires synchronous execution of shared events (events with common labels) and interleaved (independent) execution otherwise; 2) the *interleaving parallel composition operator*, denoted by |||, which allows only the interleaved execution of events. Figure 4 shows an example for each of the two parallel composition operators. Note that interleaving parallel composition can result in non-deterministic models, see Figure 4d. However, in the patient support case, interleaving does not result in a non-deterministic model. The events which are shared by the automata which are composted by interleaving parallel composition, only occur in self-loops.

4.1 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. Furthermore, infinite traces of controllable events should be prevented.

4.1.1 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:

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

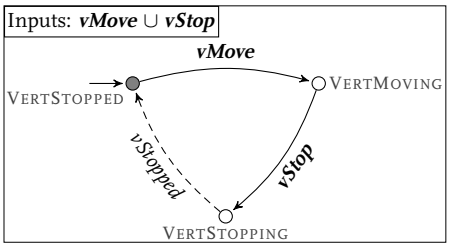
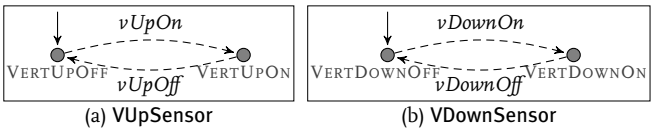


Figure 5: Model VActuators: vertical actuators



(a) VUpSensor (b) VDownSensor

Figure 6: Models VSensors: vertical sensors

The actuator, sensor and structure models are defined as:

$$\mathsf{VActuators} \triangleq \mathsf{VMotor}$$

$$\mathsf{VSensors} \triangleq \mathsf{VUpSensor} \parallel \mathsf{VDownSensor}$$

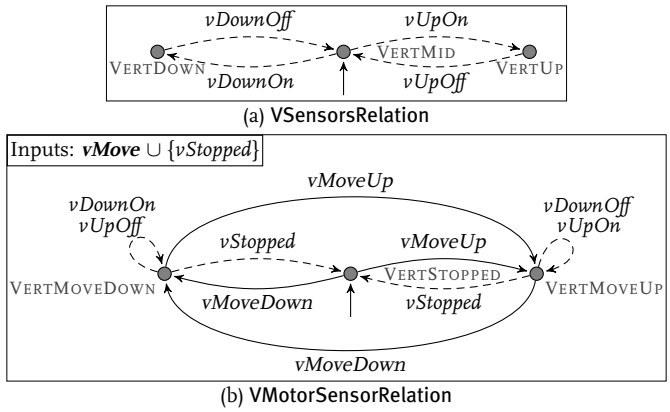
$$\mathsf{VStructure} \triangleq \mathsf{VSensorsRelation} \parallel \mathsf{VMotorSensorRelation}$$

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 5. Initially, the motor is stopped. From this state, a movement can be started (***vMove***, see Section 4.1.2 for the definition). If the motor is moving and a stop event in the set ***vStop*** (see Section 4.1.2 for the definition) 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.

Sensors (VDownSensor, VUpSensor) The maximally up and maximally down sensors are modeled in Figures 6a and 6b. 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.

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 7a. The model includes the complete behavior of the two individual sensors. Hence, the models of the individual sensors could be omitted from the plant model.

Motor-sensor relation (VMotorSensorRelation) The sensors do not change state when the table is not moving vertically, see Figure 7b. 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.



(a) VSensorsRelation

(b) VMotorSensorRelation

Figure 7: Models VStructure: vertical structure

4.1.2 Control requirements (VReq)

In Figure 5 and 7b, the events set ***vStop*** denotes a number of different of stop events, namely ***vStop*** $\triangleq$ {*vStopUp, vStopDown, vStopTTR, vStopTumble*}. This facilitates decomposition of the complete stop behavior into multiple independent requirements. The stops events should be disabled most of the time, however the 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 event set ***vMove*** is an abbreviation for the move events, ***vMove*** $\triangleq$ {*vMoveUp, vMoveDown*}.

The requirement model of the vertical axis consists of the synchronous parallel composition of multiple control requirements, namely:

$$\begin{aligned} \mathsf{VReq} \triangleq\ & \mathsf{VStopUpDown1} \parallel \mathsf{VStopUpDown2} \\ & \parallel \mathsf{VMotorSequence} \end{aligned}$$

Maximally up and down (VStopUpDown1/2) 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. Note that the table must be allowed to move up in the upper most position, otherwise this position could never be reached. Likewise it is not allowed to move beyond the maximally down position.

These two requirements are modeled together, see Figure 8a. First, the event *vMoveUp* (*vMoveDown*) is only allowed if the table is not maximally up (down). Second, the event *vStopUp* (*vStopDown*) is enabled in the maximally up (down) position. This allows the table to stop after the end position has been reached. Finally, the *vStopUp* (*vStopDown*) event is only enabled if the motor is moving up (down), see Figure 8b. The synchronizing semantics of the parallel composition in VStopUpDown1 $\parallel$ VStopUpDown2 ensures that event *vStopDown* (*vStopUp*) is enabled only if the states VertDown (VertUp) and VertMoveDown (VertMoveUp) are both active (see Figures 8a and 8b).

Loop prevention (VMotorSequence) The requirements specified so far still allow loops of controllable actuator events. For instance, the event *vMoveUp* can be executed an arbitrary number of times in succession, when enabled. To prevent such sequences of actuator events,

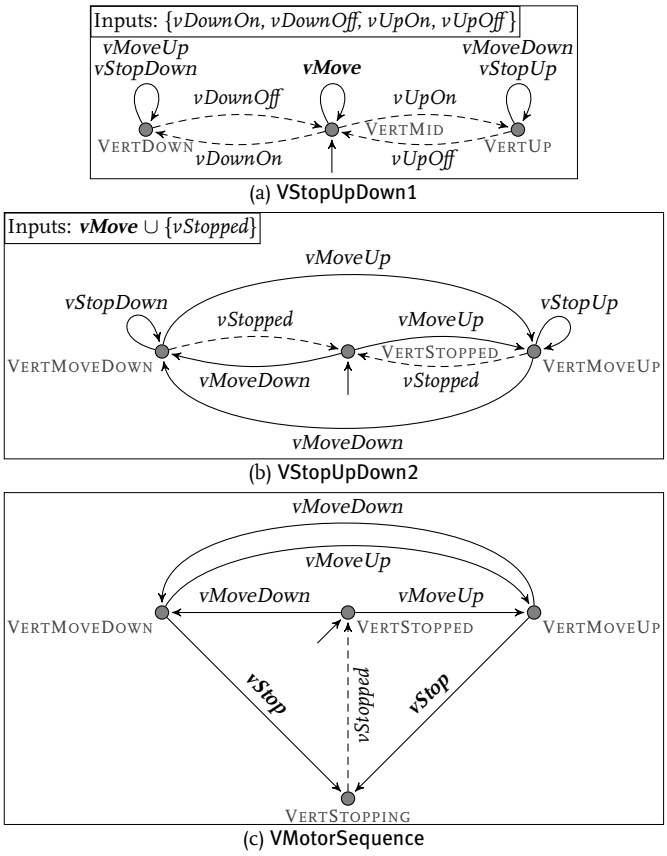


Figure 8: Model **VReq**: vertical axis requirements

requirements are added to the model. The requirement in Figure 8c disables repetitions of ***vMove*** and ***vStop*** events motor events: each move event must be followed by a stop event, or a movement in the other direction. After a stop event, the motor must come to a complete hold before another move event can be executed.

4.2 Horizontal axis

The horizontal axis consists of a removable tabletop on top of the main table. The tabletop can be moved in and out of the bore (if present). 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 TableTop Release (TTR). The clutch is released when the TTR switch is active, independent 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 release or when the TTR switch is active.

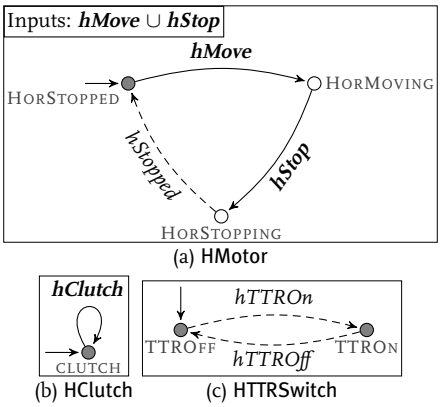


Figure 9: Model HActuators: horizontal actuators

4.2.1 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:

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

The actuator, sensor and structure models are defined as:

$$\mathsf{HActuators} \triangleq \mathsf{HMotor} \parallel \mathsf{HClutch} \parallel \mathsf{HTTRSwitch}$$

$$\mathsf{HSensors} \triangleq \mathsf{HInSensor} \parallel \mathsf{HOutSensor} \parallel \mathsf{HTabletopSensor}$$

$$\mathsf{HStructure} \triangleq \mathsf{HSensorsRelation} \parallel \mathsf{HActuatorSensorRelations}$$

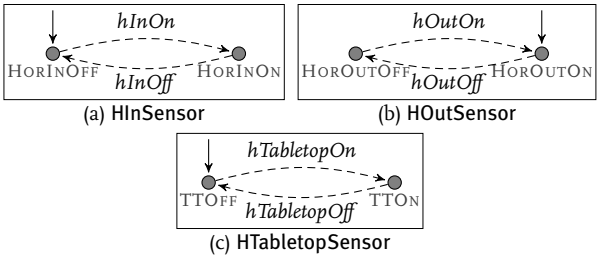
The components of HActuatorSensorRelations are composed by interleaving parallel composition:

$$\mathsf{HActuatorSensorRelations} \triangleq \mathsf{HMotorSensorRelation} \;|||\; \mathsf{HClutchSensorRelation} \;|||\; \mathsf{HTTRSensorRelation}$$

Actuators (HMotor, HClutch, HTTRSwitch) The horizontal motor drive is similar to the vertical motor drive, see Figure 9a. The clutch is modeled with one state (CLUTCH) in which the clutch events are self-looped, see Figure 9b. The tabletop release switch is modeled similarly to a sensor, see Figure 9c.

Sensors (HInSensor, HOutSensor, HTabletopSensor) The sensors of the horizontal axis are modeled similarly to the sensors in the vertical axis, see Figures 10a–c. Initially, the tabletop is not present (TTOFF), the maximally out sensor is on (HOUTON), and the maximally in sensor is off (VMAXINOFF).

Sensors relations (HSensorsRelation) The two end-sensors cannot be active at the same time, as result of their physical location, see Figure 11. Furthermore, the tabletop can only be added and removed in the maximally out position. Note that this model includes the complete behavior of the individual sensors. Hence, the models of the individual sensors could be omitted from the plant model.



(a) HInSensor (b) HOutSensor

(c) HTabletopSensor

Figure 10: Model HSensors: horizontal sensors

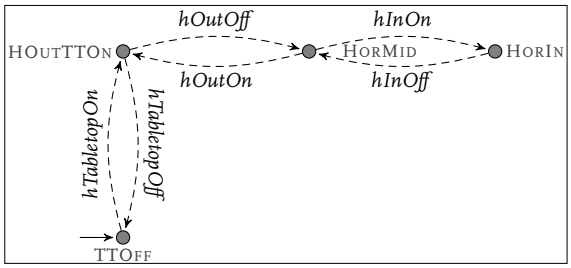


Figure 11: Model HSensorsRelation: horizontal structure (synchronizing parallel composition part)

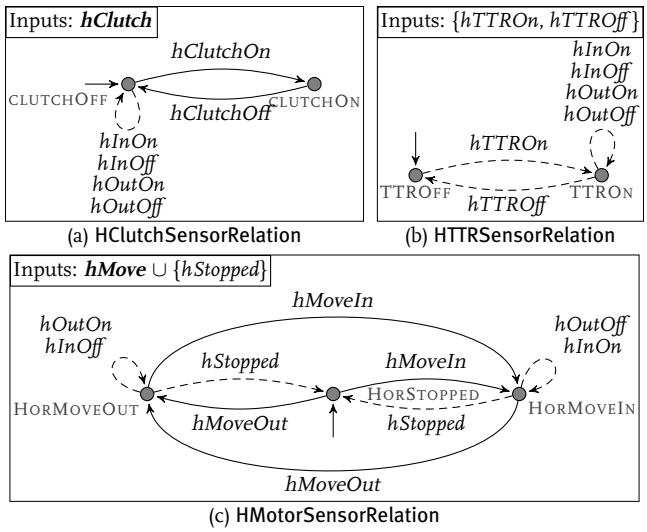
Sensor-actuator relation (HActuatorSensorRelations) Only when the tabletop is present and is moving horizontally, the maximally in and out sensors can change state. The tabletop can move horizontally in three distinct cases:

- The clutch is released, see Figure 12a; the table can be moved by hand, therefore the sensors can always switch in any order. Note that the state of the clutch is defined by the control system.
- The TTR switch is activated, see Figure 12b; the table can be moved by hand, as in the case that the clutch is released. Note that, the state of the TTR switch is defined by the operator.
- If the clutch is applied, and the TTR switch is not active, the movement is controlled by the motor, see Figure 12c; analogous to the vertical axis.

These three cases can be considered as an OR relation. Synchronizing parallel composition can be considered to model an AND relation. Therefore this form of parallel composition cannot be used to model these cases independently. Using interleaving parallel composition, an OR relation can be modeled. An equivalent automaton without parallelism would consist of $2 \times 2 \times 3 = 12$ states. I.e. HActuatorSensorRelations $\triangleq$ HMotorSensorRelation $|||$ HClutchSensorRelation $|||$ HTTRSensorRelation.

4.2.2 Control requirements (HReq)

The event set ***hStop*** denotes a number of different stop events, namely ***hStop*** $\triangleq$ {*hStopIn, hStopOut, hStopTTR, hStopTabletop, hStopTumble*}. The event set ***hClutch*** is an abbreviation for the clutch events, ***hClutch*** $\triangleq$ {*hClutchOn, hClutchOff*}, and the event set ***hMove*** is an abbreviation for the move events, ***hMove*** $\triangleq$ {*hMoveIn, hMoveOut*}.



(a) HClutchSensorRelation (b) HTTRSensorRelation

(c) HMotorSensorRelation

Figure 12: Model HActuatorSensorRelations: Actuator sensor relation (interleaving parallel composition)

The requirement model of the horizontal axis consists of the parallel composition of multiple control requirements, namely:

$$\begin{aligned}\mathsf{HReq} \triangleq\ & \mathsf{HStopInOut1} \parallel \mathsf{HStopInOut2} \\ & \parallel \mathsf{HStopTabletop} \parallel \mathsf{HClutchMove} \\ & \parallel \mathsf{HStopTTR} \parallel \mathsf{HClutchSequence} \\ & \parallel \mathsf{HMotorSequence}\end{aligned}$$

Maximally in and out (HStopInOut1/2) The horizontal axis may not move beyond its maximally in and out position, see Figures 13a and 13b.

Tabletop move (HStopTabletop) When the tabletop is not present, initiating horizontal movement is not allowed and the motor should be stopped, see Figure 13c. It is not possible to prevent the table from moving horizontally without tabletop, because the tabletop can be removed by the operator while the table is moving horizontally. In other words, the supervisor cannot ensure that the table is not moving when the operator removes the tabletop.

Clutch move (HClutchMove) The tabletop may only be moved by the horizontal motor if the clutch is applied, see Figure 13d. 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.

TTR move and clutch (HStopTTR) Commands for horizontal movement and clutch commands may only be issued when the TTR switch is off, see Figure 13e. 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*).

Loop prevention (HClutchSequence, HMotorSequence) Figure 13f prevents sequences of the same clutch events, Figure 13g prevents sequences of the same motor events.

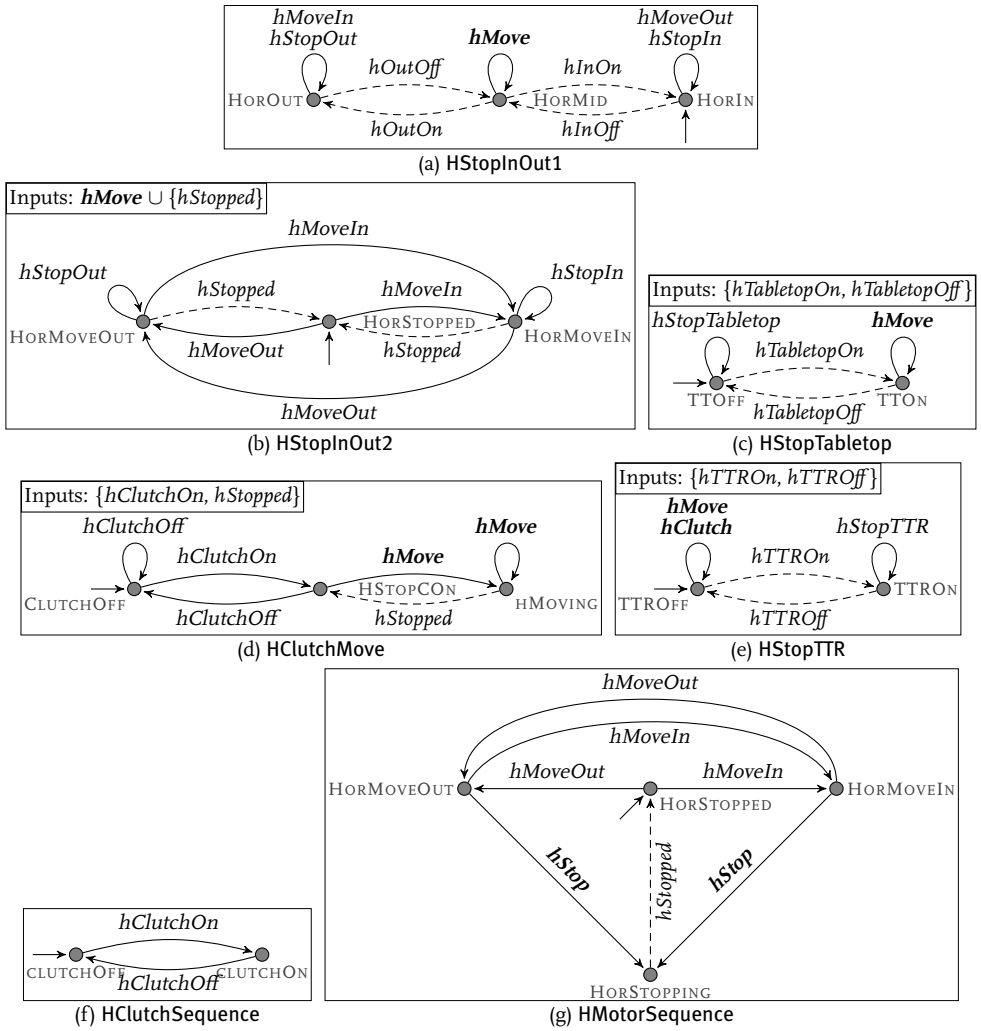


Figure 13: Model HReq: horizontal control requirements

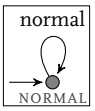


Figure 14: Model HVNormal: control modes

4.3 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.

4.3.1 Plant model (HVNormal)

In Figure 14 the internal event *normal* is introduced. This event is used to distinguish two control modes in the requirements, namely, 1) control after TTR is activated, and 2) control after the normal event has occurred.

4.3.2 Control requirements (HVReq)

The control requirements for horizontal and vertical interaction are defined by the synchronizing parallel composition of two models:

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

Movement restrictions (HVMove) Initially the system is in the state RESTRICTED, see Figure 15a. 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.

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 15b. Normal mode is represented by the states VHN, VHN and VHN. 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. V represents not vertically maximally up. After an event *hTTROn*, any horizontal or vertical position can be reached (corresponding to the states VHN, VHN, VHN and VHN).

Notice that in Figure 15b state VHN is not present. The control requirement forbids that this state is 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.

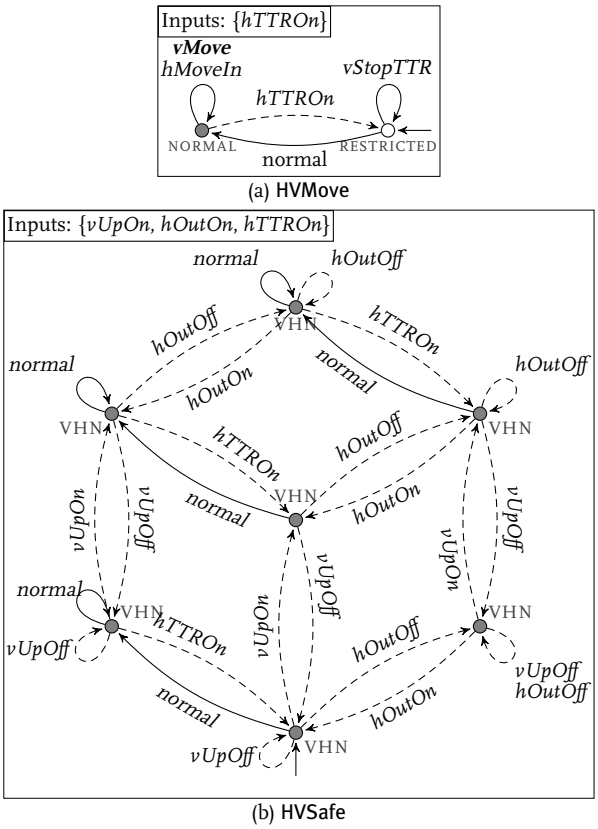
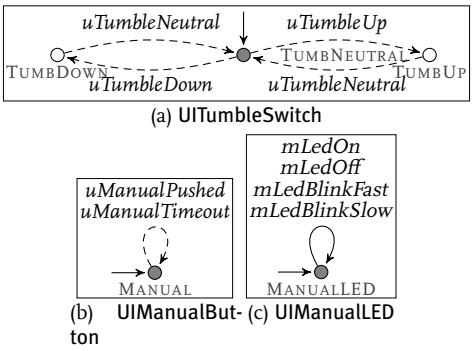


Figure 15: Model **HVReq**: horizontal and vertical restrictions



(a) UITumbleSwitch

(b) UIManualButton

(c) UIManualLED

Figure 16: Model UI: user interface

4.4 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 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.

4.4.1 Plant model (UI)

The user interface only contains actuators. The button and switch generate uncontrollable actuator events. The plant model of the user interface model consists of the synchronous parallel compositions of the actuators:

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

Tumble switch (UITumbleSwitch) The tumble switch can either be in the position up, down, or neutral, see Figure 16a. When released, the switch returns to the neutral state, as a result of its physical construction. Therefore only state NEUTRAL is marked.

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. An infinite sequence of rapid presses of the manual button could thus lead to an infinite model. This behavior is simplified to one state where the two events are self-looped, see Figure 16b.

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 16c.

4.4.2 Control requirements (UIReq)

The requirement model for the user interface consists of the parallel composition of multiple control requirements, namely:

$$\text{UIReq} \triangleq \text{UITumbleMove} \parallel \text{UIHVSwitch} \parallel \text{UIManualClutch} \parallel \text{UILedModes} \parallel \text{UILedClutch} \parallel \text{UILedSequence}$$

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 17a.

Tumble hv switch (UIHVSwitch) If the table is moving up and reaches the upper most 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 17b.

Manual clutch(UIManualClutch) Pushing the manual button should trigger an event *hClutchOn* or *hClutchOff*, if one of these events is allowed by the other requirements. If both *hClutchOn* and *hClutchOff* are not allowed, the push event is ignored when a timeout occurs, see Figure 17c.

LED (UILedModes, UILedClutch) The LED indicates which operation 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 17d. If the clutch is applied, the LED is off or blinks slow, see Figure 17e. If the clutch is released, the LED is on or it blinks fast.

Loop prevention (UILedSequence) In a way similar to other actuators, repetitions of the events that control the LED must be prevented, see Figure 17f.

5 Generation, implementation and validation of the control software

The model of the complete uncontrolled patient support system (which is referred to as the plant model) is defined as the following synchronous parallel composition:

$$\text{VAxis} \parallel \text{HAxis} \parallel \text{HVNormal} \parallel \text{UI}$$

This model consists of 7.200 states and 190.920 transitions.

The model of the complete control system requirement is obtained in a similar way:

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

This specification consists of 75.520 states and 997.448 transitions.

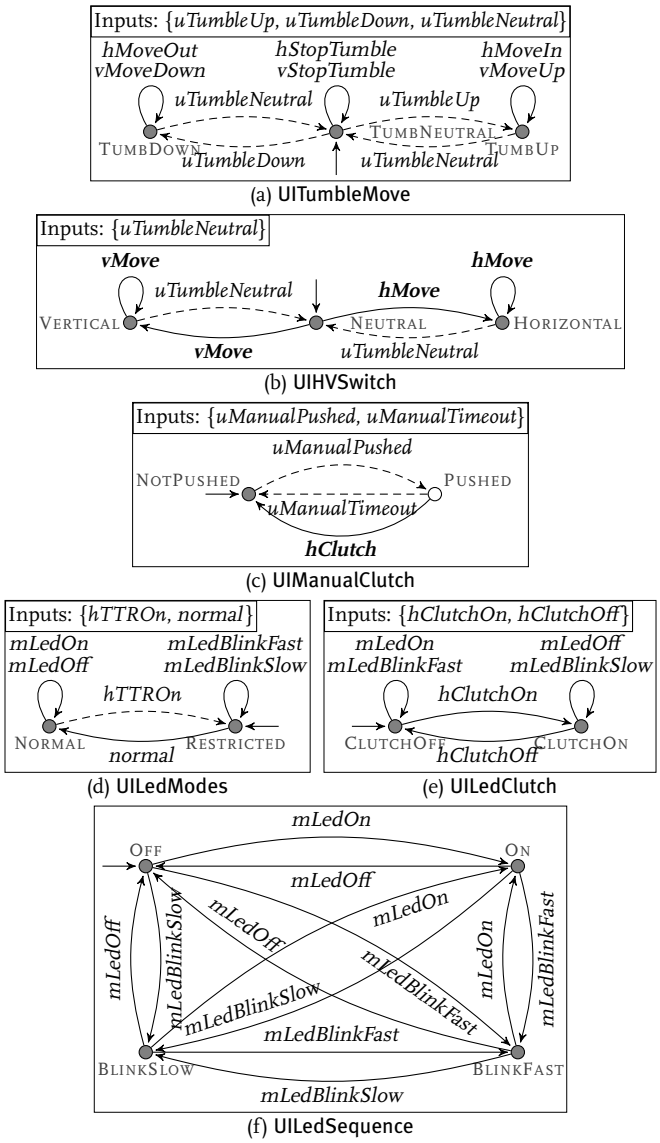


Figure 17: Model UIReq: User interface requirements

From these two models, the supervisor is generated using a Python implementation of the algorithms described in [28]. Calculation takes about a minute on a Core Duo, 2Ghz, 2Gb computer. The resulting supervisor contains 31.128 states and 265.364 transitions.

5.1 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 validating the controlled system, in [25], a framework has been developed to support the supervisory controller design process. It is based on the model-based engineering paradigm, where models are the primary artifacts in the design process. The design process used in this case study consist 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 (untimed) of the uncontrolled plant models from step A controlled by 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 plant models from step C controlled by the supervisor obtained in step B.

F) Real-time, model based simulation of the plant hardware controlled by 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, and
  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 are based on the Common Interchange Format (CIF), see [4, 3], [27], [2]. Steps C, E, and H are described in more detail in the sections below.

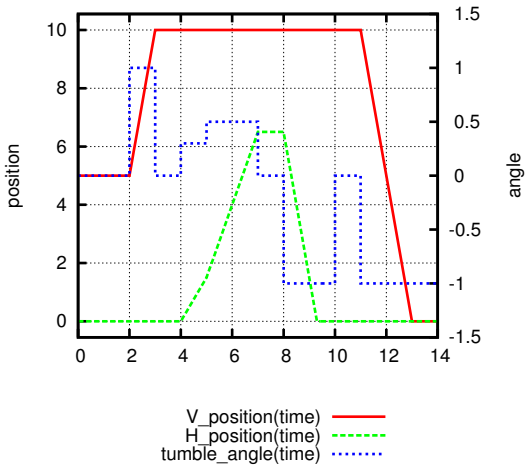


Figure 18: Simulation results S/P_{HY}.cif.

5.2 Untimed simulation

To validate the synthesized supervisor, the state space of the model of the *controlled system* is explored by hand, using user guided simulation. The model of the controlled system consists of the synchronizing parallel composition of the plant model and the synthesized supervisor. Note that, as a result of the used synthesis algorithms, the synthesized supervisor includes the plant behavior. Therefore, the behavior of the supervisor is equal to the behavior of the supervisor in parallel with the plant.

Based on different scenarios, events are chosen manually and executed. In each state, the enabled events (both controllable and uncontrollable) are inspected. For each event it is determined whether or not it is expected to occur in this state. If an inconsistency between the modeler's expectation and the synthesized supervisor occurs, it is investigated whether there is an error in the models, or the expectation of the modeler is incorrect. In this way the plant model and control requirements are validated.

5.3 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 and use cases. The hybrid model of the plant is specified in the CIF, modeling both discrete-event and continuous time behavior. Furthermore, a use case is included in the hybrid model.

Figure 18 shows the simulation results of the following use case. Initially, the table is halfway up, and the tabletop is added at the maximally out position. The tumble switch is used to move the table to the upper position ($\mathit{time} = 2, \mathit{tumble\_angle} = 1$). When the table reaches the upper position ($\mathit{time} = 3$), the table stops, and the tumble switch is released ($\mathit{tumble\_angle} = 0$). Then the table is moved inward, first slowly ($\mathit{time} = 4, \mathit{tumble\_angle} = 0.3$), then faster ($\mathit{time} = 5, \mathit{tumble\_angle} = 0.5$). After that, the movement is stopped ($\mathit{time} = 7, \mathit{tumble\_angle} = 0$). Then, the table is moved out ($\mathit{time} = 8, \mathit{tumble\_angle} = -1$), until the table reaches the maximally out position and it stops ($\mathit{time} = 9$). The tumble switch is released again ($\mathit{time} = 10, \mathit{tumble\_angle} = 0$). Finally, table is moved down ($\mathit{time} = 11, \mathit{tumble\_angle} = 0$) until it reaches the lowest position.

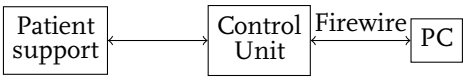


Figure 19: Implementation setup

5.4 Real-time implementation of the controller

Supervisory control theory assumes that the supervisor and the plant interact in a synchronous manner. That is, the plant makes a state-transition only when the supervisor does, and vice-versa. This synchronous interaction is also reflected in the previously described simulation models. However, in the real-time implementation, the supervisor generates controllable events and sends them to the plant. The details are as follows.

The sensors and actuators of the actual patient support table are connected to an industrial grade control unit. This control unit is connected to a standard PC by means of a firewire connection, see Figure 19. The control unit conditions the sensor signals, and takes care of motion and I/O control. Furthermore, it translates sensor state changes to events and it executes the high-level commands generated by the PC. On the PC, the events from the control unit are buffered in an event queue, and handled by an event handler. After receiving an event from the control unit, 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 non-deterministically and sent to the control unit for execution.

Note that time passes between the occurrence of an uncontrollable event in the plant, and the arrival of the response of the controller at the plant. More precisely: after a change of state of a sensor, this change has to be detected by the I/O controller (sensor polling delay). Then the I/O controller sends an event to the PC (communication delay between I/O controller and PC). After detection of the event (event queue polling delay), the state of the supervisor is updated, the allowed events are calculated, and an event is selected (calculation delay).

Hence, in principle, the real plant could perform a state-transition triggered by an uncontrollable event, while at the same time a controllable event is underway from the supervisor to the plant. If this happens, then the state of the supervisor will not correctly reflect the state of the plant. However, in the setup, first all events from the event queue are processed. Then, when the event queue is empty, an allowed controllable event is selected and sent to the I/O controller. This ensures that all uncontrollable events that have been generated by the plant and have been received by the controller during the calculation of the controller response, are taken into account. Furthermore, the delay in communication between PC, the I/O controller and the actual hardware is in the order of a few milliseconds. Therefore, when a controllable event is sent to the I/O controller, it is unlikely that the plant generates an uncontrollable event before the controllable event is processed.

6 Evaluation of evolvability

The goal of this section is to present the experimental results on the effect of using supervisory control theory on evolvability of the controller. The experiment involved generating a new controller in order to meet a user request for improved functionality. Note that for the actual experiment, models are used which are slightly different from, but functionally equivalent to, the models presented above.

The original control requirements resulted in a controller such that if the table is not maximally up and not maximally out, there is no movement of the table when the user switches the tumble switch up. 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 first move out, and when the table had reached the maximally out position, the table should move up (if the tumble switch is still up).

To model this requirement, the event *hMoveOut* is changed to denoting two different events: $\boldsymbol{hMoveOut} \triangleq \{hMoveOutNor, hMoveOutRest\}$. Furthermore, in Figure 15a, the *hMoveOutNor* is added as a self-loop in state NORMAL, and event *hMoveOutRest* is added as a self-loop in state RESTRICTED. Finally, in Figure 17a, the event *hMoveOut* is replaced by *hMoveOutNor*, and the event *hMoveOutRest* is added to the self-loops in states TUMBLEDOWN and TUMBLEUP. This new requirement has been implemented on the actual patient support table four hours after it was conceived. Implementing the same requirement using the currently used design approach is estimated to take a week.

7 Conclusions and future work

In this paper, supervisory control theory has been used to synthesize a supervisory controller for a patient support system of an MRI scanner. The use of supervisory control enables easy adaptation to changing control requirements. In the case of a change, the only thing that needs to be done manually is to formalize the new requirements. After the formalization step is completed, the theory and tools provided by supervisory control framework allow automatic generation of control software.

This paper presents first results on the use of SCT for improving evolvability of high-tech systems. Much more work needs to be done to explore the scope of applicability of SCT for this purpose. In particular, our research focuses on the following issues.

- **Further case-studies** More industrial case-studies are carried out in order to further evaluate the applicability of the approach.
- **Modular supervisory control** The usefulness of modular supervisory control for the system presented in the paper is investigated.
- **State based requirements and models** In this paper classical event-based supervisory control theory is applied. In event-based supervisory control the control requirements are formulated as the set of sequences which the closed-loop system is allowed to generate. However, the actual control requirements often demand that the closed-loop system never enters a certain state or that certain events are not generated if the closed-loop system is in a certain state. Expressing such *state-based control requirements* in classical event-based supervisory control is cumbersome. In addition, the plant model often has a hierarchical structure, where the states of the plant consist of several components. The presence of state-based control requirements and hierarchical structure lead us to think that *state-based supervisory control theory* as defined in [19] might be more suitable for control of certain high-tech systems. For this reason, the application of state-based supervisory control theory to the case-study of the paper and to other case-studies is investigated. For preliminary results, see [5].
- **Initialization** Classical supervisory control assumes that the plant has one initial state. However, in practice the plant may have several possible initial states. Developing a controller for each initial state separately is not very practical. For this reason, the

possibility of adapting the supervisory control synthesis procedure is investigated so that several initial states would be allowed.

- **Error behavior** The plant model of this paper describes the behavior of the plant in the absence of errors. However, error-handling is one of the most challenging control problems for high-tech systems. Modeling and control of the error behavior of high-tech systems gives rise to a number of research questions. One of them is how to model the error behavior of the plant. While it is always possible to incorporate the error behavior into the plant model, it is not at all desirable to build monolithic plant models which attempt to incorporate all the error behavior. Rather, one would like to develop systematic methods to add error behavior to the models in a modular way. This calls for methods for modular modeling of error behavior. For preliminary results see [5].
- **Asynchronous communication between plant and supervisor** Supervisory control theory assumes that the supervisor and the plant interact in a synchronous manner. That is, the plant makes a state-transition only when the supervisor does, and vice-versa. However, this is usually not the case, see Section 5.4. Hence, the question arises whether the closed-loop system functions correctly, if the assumption of synchronous communication is dropped. Note that the specific features of the case-study, as explained in Section 5.4, imply that asynchronity is not likely to cause incorrect behavior of the closed-loop system. For other systems, this need not be the case.

Acknowledgment The authors thank Albert Hofkamp and Dennis Hendriks for their contribution to the development of the software tools. The authors thank Rong Su for his help with the theoretical aspects of SCT.

Bibliography

[1] S. Balemi, G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G.F. Franklin. Supervisory control of a rapid thermal multiprocessor. *Automatic Control, IEEE Transactions on*, 38(7):1040–1059, 1993.

[2] D. A. van Beek, P. Collins, D. E. Nadales, J.E. Rooda, and R. R. H. Schiffelers. New concepts in the abstract format of the compositional interchange format. In A. Giua, C. Mahuela, M. Silva, and J. Zaytoon, editors, *3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 250–255, Zaragoza, Spain, 2009.

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

[4] D. A. van Beek, M. A. Reniers, R. R. H. Schiffelers, and J. E. Rooda. Foundations of an interchange format for hybrid systems. In Alberto Bemporad, Antonio Bicchi, and Giorgio Butazzo, editors, *Hybrid Systems: Computation and Control, 10th International Workshop*, volume 4416 of *Lecture Notes in Computer Science*, pages 587–600, Pisa, 2007. Springer-Verlag.

[5] E. Bertens, R. Fabel, M. Petreczky, D.A. van Beek, and J.E. Rooda. Supervisory control synthesis for exception handling in printers. In *In Proceedings Philips Conference on Applications of Control Technology*, 2009.

[6] B. Brandin and F. Charbonnier. The supervisory control of the automated manufacturing system of the AIP. *Proc. Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 319–324, 1994.

[7] B.A. Brandin. The real-time supervisory control of an experimental manufacturing cell. *Robotics and Automation, IEEE Transactions on*, 12(1):1–14, 1996.

[8] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2007.

[9] Vigyan Chandra, Zhongdong Huang, and Ratnesh Kumar. Automated control synthesis for an assembly line using discrete event system control theory. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 33(2):284–289, 2003.

[10] Gde.O. Costa, B. Tortelli, E.A.P. Santos, and M.A. Busetti. Design and implementation of a low cost control system for a manufacturing cell. In *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, volume 1, pages 281–286 vol.1, 2004.

[11] M.H. de Queiroz and J.E.R. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 377–382, 2002.

[12] Í. Hasdemir, Salman Kurtulan, and Leyla Gören. An implementation methodology for supervisory control theory. *The International Journal of Advanced Manufacturing Technology*, 36(3):373–385, March 2008.

[13] A. Hellgren, M. Fabian, and B. Lennartson. Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*, pages 453–458, 2001.

[14] Sangkyun Kim, Jinwoo Park, and Robert C. Leachman. A supervisory control approach for execution control of an FMC. *International Journal of Flexible Manufacturing Systems*, 13(1):5–31, February 2001.

[15] J. Kosecka and L. Bogoni. Application of discrete events systems for modeling and controlling robotic agents. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 2557–2562 vol.3, 1994.

[16] S.C. Lauzon, A.K.L. Ma, J.K. Mills, and B. Benhabib. Application of discrete-event-system theory to flexible manufacturing. *Control Systems Magazine, IEEE*, 16(1):41–48, 1996.

[17] R.J. Leduc and W.M. Wonham. Discrete event systems modeling and control of a manufacturing testbed. In *Electrical and Computer Engineering, 1995. Canadian Conference on*, volume 2, pages 793–796 vol.2, 1995.

[18] Jing Liu and Houshang Darabi. Ramadge-Wonham supervisory control of mobile robots: lessons from practice. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 670–675, 2002.

[19] C. Ma and W. M. Wonham. *Nonblocking supervisory control of state tree structures*, volume 51 of *IEEE Transactions on Automatic Control*. IEEE, 2006.

[20] M. Moniruzzaman and P. Gohari. Implementing supervisory control maps with PLC. In *American Control Conference, 2007. ACC '07*, pages 3594–3599, 2007.

[21] M. Noorbakhsh and A. Afzalian. Design and PLC based implementation of supervisory control for under-load tap-changing transformers. In *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, pages 901–906, 2007.

[22] M. Nourelfath and E. Niel. Modular supervisory control of an experimental automated manufacturing system. *Control Engineering Practice*, 12(2):205–216, February 2004.

[23] Jean-Fran cois Pétin, David Gouyon, and Gérard Morel. Supervisory synthesis for product-driven automation and its application to a flexible assembly cell. *Control Engineering Practice*, 15(5):595–614, May 2007.

[24] E.A. Santos, M.A. Busetti, and A.D. Vieira. Control synthesis and implementation for an integrated manufacturing system based on supervisory control theory. In *Control Applications, 2006. CCA '06. IEEE International Conference on*, pages 1885–1890, 2006.

[25] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda. Model-based engineering of supervisory controllers using CIF. *Electronic Communications of the EASST*, 21:1–10, 2010.

[26] Kiam Tian Seow and M. Pasquier. Supervising passenger land-transport systems. *Intelligent Transportation Systems, IEEE Transactions on*, 5(3):165–176, 2004.

[27] C. Sonntag, R. R. H. Schiffelers, D. A. van Beek, J. E. Rooda, and S. Engell. Modeling and simulation using the Compositional Interchange Format for hybrid systems. In I. Troch and F. Breitenecker, editors, *6th International Conference on Mathematical Modelling*, Vienna, Austria, 2009.

[28] R. Su, J.H. van Schuppen, and J.E. Rooda. Aggregative synthesis of distributed supervisors based on automaton abstraction. *Automatic Control, IEEE Transactions on*, page accepted, 2009. It also appears in SE Technical Report No. 2009-1, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Techonology, Eindhoven, The Netherlands, 2009. ISSN 1567-1872. URL: http://se.wtb.tue.nl/sereports.

[29] R. J. M. Theunissen, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda. Supervisory control synthesis for a patient support system. SE Report 2008 – 08, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2008.

[30] 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.

[31] W.M. Wonham. *Supervisory control of discrete-event systems*. Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2007.