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. 2008-08

Supervisory control synthesis for a patient support system

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

ISSN: 1872-1567

SE Report: Nr. 2008-08
Eindhoven, December 2008
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

The increasing complexity of systems and the increasing market pressure necessitate the need for methods to maximize reuse and to minimize the effort to develop new systems. Model-based engineering is one of these methods. It uses models and model-based techniques in the development process to analyze and synthesize systems and components.

In this report, Supervisory Control Synthesis is used to design a supervisory controller for a patient support system. This system is used to position a patient in a Magnetic Resonance Imaging (MRI) scanner. To improve the evolvability of the design, the uncontrolled system and the control requirements are modeled independently, using small loosely coupled minimal restrictive automata. An implementation of the synthesized supervisor is realized by means of a transformation to an automaton in the Compositional Interchange Format (CIF). The supervisor is validated by means of hardware-in-the-loop simulation, using the real patient support system.

Contents

Chapter 1
Introduction

High-tech systems, such as wafer scanners or MRI scanners, are usually not developed from scratch. They have evolved during multiple generations of the system, from relatively simple systems into highly advanced systems. Developing new generations of these systems is challenging, due to market demands and increased competition. The number of (advanced) features in these systems is increasing. Thus the complexity of these systems increases. The new features should be developed in less time, because the time-to-market of the systems decreases. At the same time, the new systems must meet high quality constrains. This necessitates the need for methods to maximize reuse and minimize the effort to develop new generations of a system.

A possible method to support the development of evolving systems is model-based engineering. In model-based engineering, models and model-based techniques are used in the development process. Different techniques can be used for analysis or synthesis. In analysis, models are used to check correctness of the behavior of a system. In supervisory control synthesis, models are used to develop a supervisory control system which is correct by construction.

In Chapter 2, this report describes the current engineering process. Furthermore, it describes the model based engineering process for both analysis and synthesis. In Chapter 3, the concept of supervisory control is discussed. In Chapter 4, the synthesis method for supervisory control is applied to a patient support system.

Chapter 2

Model-based engineering

This chapter introduces the model-based engineering approach. First the current development process is explained. After that the model-based engineering approach is introduced. The later approach contains two topics, namely analysis and synthesis, each having a different purpose and a different development process.

1 Current system development process

In current industrial practice, the system development process is usually based on a 'divide and conquerer' strategy. The system to be developed is decomposed into smaller parts which are developed separately. When all parts are available, the parts are combined to construct the system. Then, the system is tested to check if it functions correctly. The system decomposition can contain multiple levels of hierarchy. However, for simplicity, only two levels of hierarchy are considered in this discussion. The higher level is referred to as the system, the lower level is referred to as the components. Figure 2.1 shows a graphical representation of the current development process. The arrows depict different development phases, the boxes depict the different representations of the system and the components.

In the initial phase, the requirements of the system are defined. Requirements (R) define the functionality that a system or component should provide. Typically, the requirements also include constrains, for instance on performance or safety, that should be satisfied by the system as well.

In the next phase, a design of the system is made. The design (D) specifies how the system should be built to satisfy the requirements. The design specifies for example the architecture, the decomposition of the system, the internal behavior, and the technologies used. Furthermore, It specifies the interaction and interfaces between the components of the system.

The decomposition in the design defines the requirements for the components (R_i). The higher level requirements and design decisions are translated into the corresponding requirements for the components. The corresponding designs of the components (D_i) specify how the

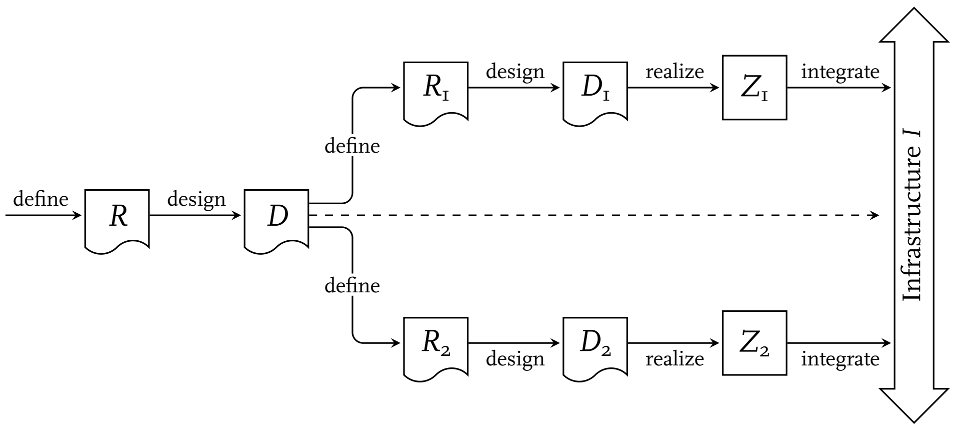


Figure 2.1: Current system development process

components satisfy these requirements. Then the components are realized according to the design, resulting in a realization of a component (Z_i). Realizations are the real components, e.g., mechanics, electronics, software, of the system. They should satisfy the requirements as defined for each component. When the realizations of all components are ready, they are integrated by means of an infrastructure (I).

When the integrated system is available, it can be tested against the system requirements (R) and the intended system design (D). Testing involves defining tests for the considered aspects of the system, executing these tests on the systems realization, and determining the test outcome (pass or fail) by comparing the test results to the requirements and designs. When problems are detected and need to be fixed, the designs or implementations must be fixed.

In this discussion the development process is simplified. In practice, the different phases need not be sequential; one phase can start before the previous one has ended. Furthermore, the development process has an incremental and iterative nature, resulting in different versions of requirements, designs and implementations, and feedback loops between phases.

2 Model-based engineering

In the current development process, three classes of system and components representations are used, namely requirements, designs and realizations. The requirements and designs are mostly based on (informal) documents only. The realizations are real components. Thus, the current development process is mostly based on documents. This has several disadvantages. First, documents may contain ambiguous or inconsistent information. Furthermore, practical experience shows that documents may be incomplete, or outdated. As a result, it is difficult to obtain a good system overview, and to detect inconsistencies and potential problems based on documents only. Second, due to the informal structure of documents, it is hard to process them automatically. This complicates automated processing analysis techniques such as inconsistency detection. This currently leaves manual document reviewing as the main technique used for document analysis. Third, documentation is a static piece of information, which makes it difficult to express and analyze the dynamic system behavior. Executable specifications [1], on the other hand, enable a more thorough and systematic analysis of dynamic system behavior. Fourth, determining the integrated system behavior based on

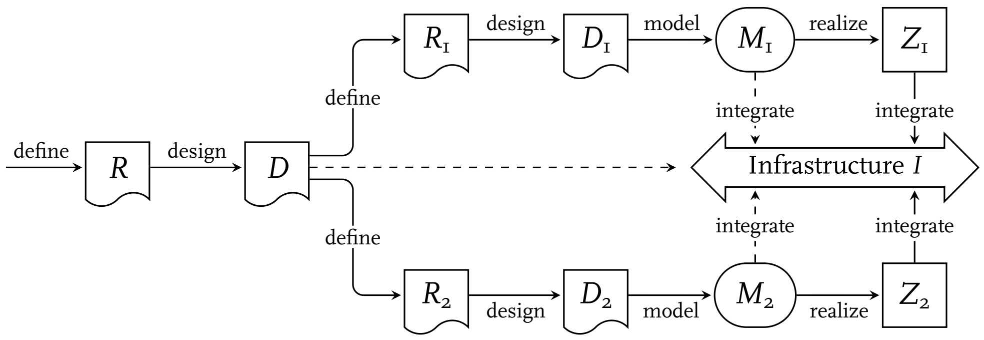


Figure 2.2: Model-based engineering development process (analysis)

component documentation only, is a difficult task. It requires a considerable amount of component design knowledge, as well as good system overview.

In other words, documentation alone is not well suited to check the correctness of the system to be built. In current development, this leaves the realizations of the components and the system as the only representations that can be checked for correctness (tested). As a result, the correctness of the system or component can only be validated, after it has been realized.

To overcome some drawbacks of the document-based development process, models can be included in the development process. Models are an abstract representation of a real component or system, used in experiments to gain knowledge about the real component or system. Although different types of models can be used, for example scale models of cars to analyze aerodynamics, we particularly focus on models describing the behavior of components and systems. Including models in the development process enables the use of a range of model-based techniques and tools to support the development process.

The main difference between models and documents is that the constructs used in models have semantics which define precisely what each construct means; documents do not have such semantics. This makes models more consistent and less ambiguous than documents. Furthermore, the well-defined model semantics enables automatic processing by tools. This enables the use of various sophisticated and automated analysis and synthesis techniques.

2.1 Model-based analysis

Model-based analysis is used to check the correctness of a system based on models. This analysis can be used as a supplement to the current development process. Therefore the impact of using model-based analysis on the current way of working is limited. Figure 2.2 shows a graphical representation of a model-based engineering process for analysis. The initial definition of the requirements and design remains the same as in the current development process. After the design of the components is completed, not only an implementation of this design is made, but also a model of the design (M_i). This allows for early validation or verification of the design.

However, since the models describe the designed behavior of a component, the integration of the models may still result in incorrect system model behavior. For instance, due to unforeseen conflicts in component designs.

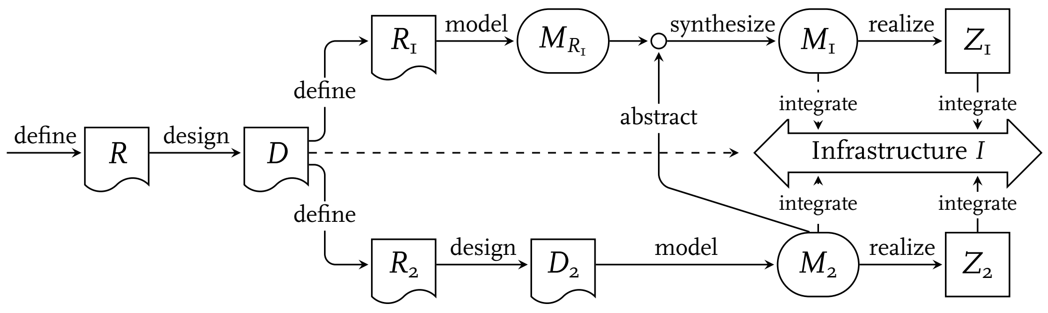


Figure 2.3: Model-based engineering development process (synthesis)

2.2 Model-based synthesis

Model-based synthesis is used to *generate* a system or component which has certain predefined properties. As a consequence, the development process changes from implementing and debugging the design and the implementation, to designing and debugging the requirements. This allows for a faster incorporation of requirement modifications into the implementation. In turn, this leads to a reduction in the number of design-test-redesign loops.

Figure 2.3 shows a graphical representation of the model-based engineering process for synthesis. In this approach, a model of the requirements is made (M_{R_1}), instead of a model of the design. This model is used, together with an abstraction of model M_2, to synthesize a model of the component (M_1). The synthesized model (M_1) can be used to automatically generate the implementation (Z_1).

Chapter 3
Supervisory machine control

Machines consist of two parts: physical components (hardware) and control systems. The physical components provide the means of the machine, that is what the machine can do. The control systems employs those means to fulfill the machine functions, that is what the machine should do. The control systems interact with the physical components by sending information to the actuators, and reading information from the sensors. In the remainder of this report, the physical components are referred to as the plant or system.

Three types of physical components can be distinguished, namely sensors, actuators, and main structure. Sensors detect the current state of the system, for instance the current position or temperature. Actuators can change the state of the system, for instance change the position, or add energy (heat). The main structure connects the sensors and actuators, posing the interaction of the sensors and the actuators. For instance, a motor (actuator) drives a shaft (structure) which turns a switch (sensor). For example in the patient support system the base frame, the scissor and the table-top are part of the main structure. The vertical and horizontal motors, the brakes and the clutch are some of the actuators. Finally, the encoders, the limit switches and the table top release sensor, are examples of sensors.

The machine control system can be divided into two layers, see Figure 3.1:

1. Resource control, or low-level control. This includes open loop and feedback control.
2. Supervisory control, which assures that a machine correctly performs its function by determining and executing an allowed sequence of tasks on (in)dependent resources. This includes planning, scheduling and dispatching functions.

The resource control system operates in a continuous, time driven domain. The supervisory control system operates in a discrete, event driven domain. An interface between these two layers provides an abstraction of the continuous to the discrete domain. It also translates the discrete signals of the supervisory layer to continuous signals for the resource layer. On top of these two control layers is the user interface which provides the interaction with the user.

Typical resource control in the patient support system is the servo control of the motors.

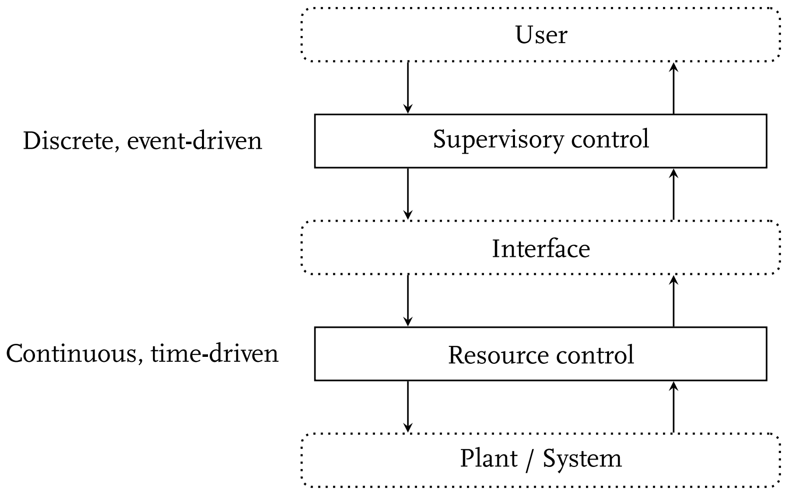


Figure 3.1: Control architecture

Typical supervisory control is applying and releasing the clutch, starting and stopping the servo control, and reacting to user input. The user interface is provided by the PICU. The remainder of this report focuses on supervisory control, it is assumed that the resource control system behaves as expected.

1 Supervisory control design

It is current practice to design discrete event controllers by hand, based on the, generally informal, requirements of the desired behavior of the system. Based on this design, an implementation of the supervisor is made. When the implementation is ready, it is integrated with the hardware and tested to validate its correctness. If it turns out that the supervisor is incorrect, the design and/or implementation are fixed to remove the errors. Then the supervisor can be tested again. This approach has several disadvantages:

- Requirements are usually ambiguous and incomplete.
- After changes in the requirements it is difficult to change the design and implementation.
- Only in the testing phase it can be validated that the right system has been built.
- Testing the system thoroughly, and finding and fixing errors is difficult, time consuming and unpredictable, potentially causing delays in market introduction.
- After the product is released there may still be errors in the product. Testing can only find errors, it cannot guarantee the absence of errors.

These disadvantages are especially true for evolving systems, in which requirements change frequently. Making new (or updated) designs and implementations is time consuming, cost intensive, and short-time goals may cause degradation of the overall quality of the system over time.

2 Supervisory control synthesis

To support the evolvability of systems, the design process of supervisory controllers can be automated. In this case, supervisory control theory (SCT), initiated by Ramadge/Wonham [2], is used. This is a model-based synthesis technique. By means of SCT, supervisors can be synthesized such that they are nonblocking and satisfy the control requirements by construction. First, the uncontrolled system (plant) and its control requirements are formally specified in terms of automata. Then, from these models, the supervisor is synthesized.

This approach has the following advantages:

1. The design of the supervisor by hand is eliminated.
2. The supervisor is correct (w.r.t. the plant model and the control requirements) by construction, which eliminates the need for exhaustive testing and verification of the supervisor implementation.
3. The models of the uncontrolled system and of the control requirements are unambiguous.
4. If the plant models and control requirements are based on small, loosely coupled specifications, changes in the plant or in the control requirements can be realized quickly, without introducing errors.
5. The synthesized supervisors are suitable for code generation. This makes the design relatively independent of the implementation technology.

The SCT is a method to synthesize supervisors for discrete event systems. The theory is based on feedback control on the observed events in the system. Depending on the observed behavior of the system, events are enabled or disabled by the supervisor, to restrict the possible next states of the system.

The uncontrolled system and the control requirements are modeled as finite state machines (automata). The automata can be composed by synchronous composition out of several smaller automata to obtain the final model of the uncontrolled system and model of the control requirements. With these two models as input, a supervisor is synthesized, which ensures that the controlled system behaves within the specification (control requirements). The resulting supervisor is also represented by an automaton.

Within SCT two classes of events are distinguished, *controllable* events and *uncontrollable* events. Controllable events (typically events of actuators) can be disabled by the supervisor, the possible occurrence of these events depend on the supervisor and the current state of the system. Uncontrollable events (typically events of sensor and user input) cannot be disabled by the supervisor, the possible occurrence of these events depend entirely on the current state of the system under control.

The control problem to solve is to restrict the behavior of the uncontrolled system such that it achieves a given set of control requirements. Note that the requirements could require that uncontrollable events must be disabled in some states, in this case the requirements are called *uncontrollable*. In such a case, those controllable events should be disabled which lead to the state in which uncontrollable events should be disabled. The generation of the supervisor is twofold:

1. Find the controllable behavior within the specification.

2. Prevent deadlock in the controlled system.

The generated supervisor within the SCT framework is *minimally restrictive* and the controlled system is *non-blocking*. Minimally restrict means that the behavior of the system is not restricted more than strictly necessary to achieve (the controllable part of) the requirements. Non-blocking means that the system is always able to finish some tasks, or to reach at least one of some predefined states, the so called marker states.

Note that the basic framework does not include time, only sequences of events can be specified. There is no notion at what time events occur, only the order in which they occur can be determined. Thought there are extensions to the framework to deal with timed specifications.

In the next chapter, the uncontrolled system and the control requirements of a patient support system are modeled. With these models a supervisor is synthesized.

Chapter 4

Patient support system

In this chapter, supervisory control theory is applied to a patient support system, see Figure 4.1. The patient support system is used to position a patient in an MRI scanner diagnosis to render pictures of the inside of a patient non-invasively. The patient support system is more difficult to control than 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).

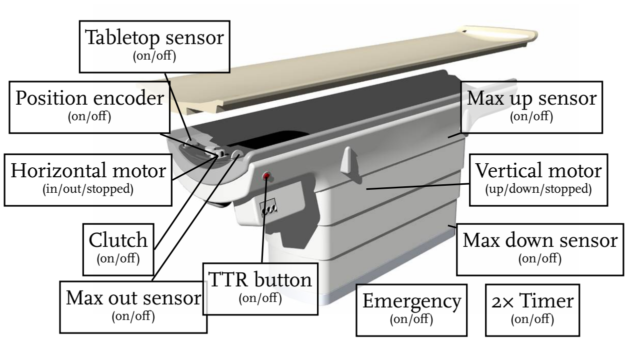


Figure 4.1: Patient support system

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

A subset of the functionality of the patient support systems is discussed in this paper. The emergency system, the light-visor with automated positioning, and the remote control system are not modeled. Also the LEDs in the local user-interface are excluded.

To ensure evolvability it is essential that plant models and control requirements of the patient support system consist of multiple models which are:

- Small and easy to understand.
- Loosely coupled, in the sense that changes in one specification lead to no changes, or to small changes in the other specifications.
- Minimally restrictive, to allow maximal freedom to specify other requirements.

1 Notations and model semantics

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 drawn with solid and dashed edges, respectively. Multiple events on a edge represent an edge for each event. In the models, and in general, the events associated to actuators are controllable, and the events associated to the sensors are uncontrollable. 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.

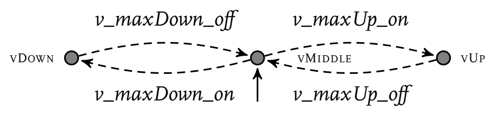
2 Vertical axis

The vertical axis contains two sensors: maximally up and maximally down, and one actuator: the vertical motor. The system should never move beyond the maximally up or maximally down position.

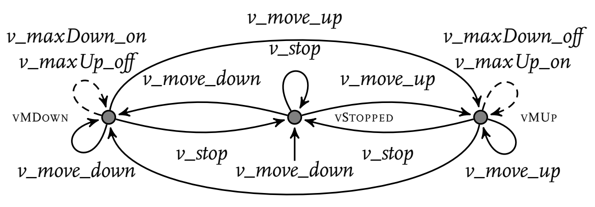
2.1 Plant model (*Vaxis*)

Initially the table is assumed to be neither up or down, so that both end sensors are inactive. Note that initialization of the table is omitted in this paper. The sensors emit the events *v_max…_on* or *v_max…_off*, when a sensor becomes active or ceases to be active, respectively (Fig. 4.2a). Because of the physical location, the sensors are never active at the same time.

The motor is initially stopped, and it always accepts its controllable events (Fig. 4.2b). The sensors do not change state when the motor is stopped. Only when the vertical motor is moving up, the maximally down sensor can turn *off* and the maximally up sensor can turn *on*, and likewise for the opposite direction.



(a) *v_sensor_positions*



(b) *v_motor_sensors*

Figure 4.2: Model *Vaxis*: vertical axis

2.2 Control requirements (*Vreq*)

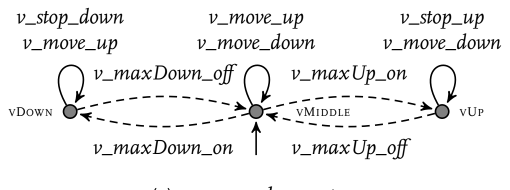
To specify the control requirements, the event *v_stop* is duplicated. The stop event *v_stop* is replaced by a number of different of stop events, namely *v_stop* $\triangleq$ *v_stop_up*, *v_stop_down*, *v_stop_TTR*, *v_stop_tumble*. This allows to model the stop behavior independently. The stop event should be disabled most of the time, however it should be enabled in distinct cases. For instance *v_stop_up* 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 vertical axis should not move beyond its maximally up and maximally down position (Fig. 4.3a). When the table is maximally up or down, it should stop (events *v_stop_up* and *v_stop_down*). In the maximally up position it is not allowed to move up (no event *v_move_up*), in the maximally down position it is not allowed to move down (no event *v_move_down*).

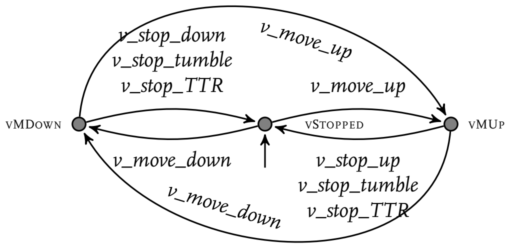
Repetitions of the controllable events are undesired in the controller. Therefore, repetitions are disabled in the control requirements. The requirement in Fig. 4.3b disables repetitions of motor events. Furthermore it disables the event *v_stop_up* when moving down, to prevent sequences of the events *v_move_down* and *v_stop_up*. Similarly, the event *v_stop_down* is disabled when moving up.

3 Horizontal axis

The horizontal axis contains four sensors: maximally in, maximally out, tabletop present and tabletop release active; and two actuators: the horizontal motor and the clutch. The tabletop can only be added and removed in the maximally out position. The clutch connects the motor to the tabletop. When the clutch is applied, the motor controls the movement of the tabletop. Otherwise the tabletop can be moved freely by the operator. The tabletop release (TTR) is a hardware safety system which releases the clutch independently of the controller. If TTR is active, the table can be moved freely, even if the controller enables the clutch.

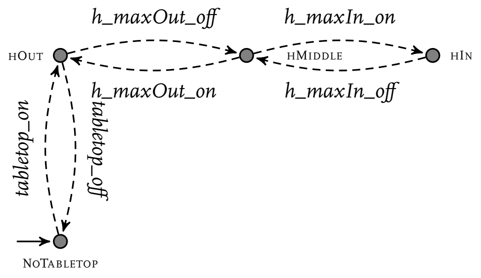


(a) *v_maxupdown_stop*



(b) *vmotor_sequence*

Figure 4.3: Model *Vreq*: vertical control requirements



(a) *h_sensor_position*

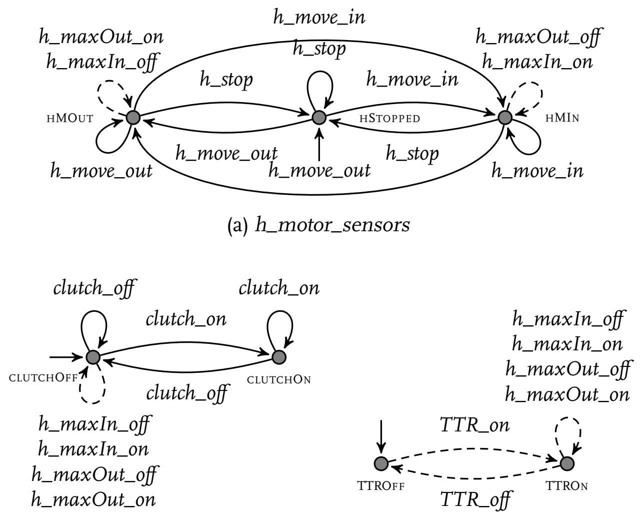
Figure 4.4: Model *HaxisA*: horizontal axis synchronizing

3.1 Plant model (*Haxis*)

The model *Haxis* of the horizontal axis is divided into two parts: $Haxis \triangleq HaxisA \parallel HaxisB$. The automata in the part *HaxisA* are composed by normal (synchronizing) parallel composition. The automata in the part *HaxisB* are composed by interleaving parallel composition.

Fig. 4.4 (*HaxisA*) models the following behavior: initially the tabletop is not present and the maximally out sensor is active. The tabletop can only be added and removed in the maximally out position (events *tabletop_on* and *tabletop_off*). When the tabletop is present the maximally out and maximally in sensors can switch, and will emit the respective *on* and *off* events. Likewise, the maximally in and maximally out sensors cannot be active at the same time.

Fig. 4.5 (*HaxisB*) models the following behavior: only when the tabletop is moving, the maximally in and out sensors can switch. The tabletop moves when it is moved by the motor drive (Fig. 4.5a, which is analogous to Fig. 4.2b). It can also be moved by an operator when the clutch is released (Fig. 4.5b) or when TTR is active (Fig. 4.5c). Because the sensors can switch in either case, these three models are composed by interleaving parallel composition, i.e. $HaxisB \triangleq h\_motor\_sensors \mathbin{|||} clutch\_sensors \mathbin{|||} TTR\_sensors$.



(a) *h_motor_sensors*

(b) *clutch_sensors*

(c) *TTR_sensors*

Figure 4.5: Model *HaxisB*: horizontal axis interleaving

3.2 Control requirements (*Hreq*)

Analogous to the event *v_stop*, the event *h_stop* is replaced to model different stop cases independently of one another: $h\_stop \triangleq h\_stop\_in, h\_stop\_out, h\_stop\_TTR, h\_stop\_tumble$.

Similarly to the vertical axis (Fig. 4.3a), the horizontal axis may not move beyond its maximally in and out position (Fig. 4.6a). In addition, when no tabletop is present, horizontal movement is not allowed and the motor should be stopped.

The tabletop may only be moved by the horizontal motor if the clutch is applied, and TTR is not active: First (Fig. 4.6b), 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. Second (Fig. 4.6c), horizontal movement is only allowed to start when TTR is off. It cannot be prevented that TTR is turned on while moving. Whenever TTR is turned on, the table should be stopped.

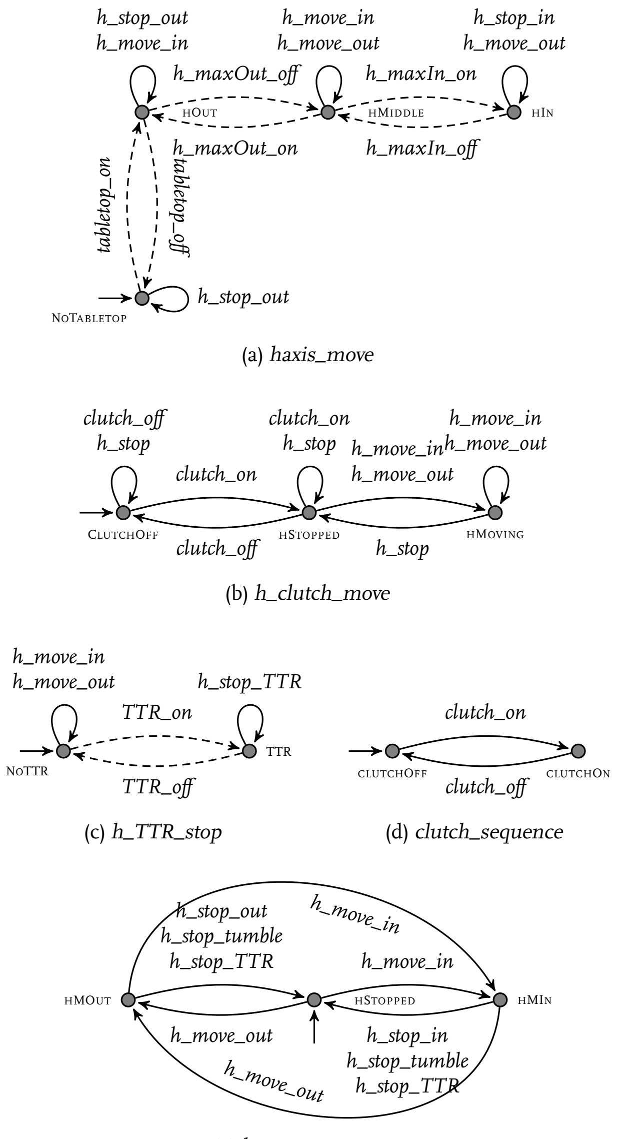
Fig. 4.6d disables repetitions of clutch events, and in analogy to the vertical axis (Fig. 4.3b), Fig. 4.6e prevents repetitions of motor events.

4 Horizontal and vertical axis

There is no physical interaction between the transducers of the horizontal and vertical axis. Therefore, no new plant models need to be introduced. Only control requirements need to be added.

4.1 Control requirements (*HVreq*)

Collisions of the tabletop with the magnet should be prevented. To accomplish this, the tabletop must be either maximally out (or not present), or the table must be maximally up. However, this condition can be violated when TTR is active. The table can then be moved



(a) *haxis_move*

(b) *h_clutch_move*

(c) *h_TTR_stop*

(d) *clutch_sequence*

(e) *hmotor_sequence*

Figure 4.6: Model *Hreq*: horizontal control requirements

freely by the operator, which may lead to a state in which the table is not maximally up and not maximally out. If TTR is not active and the table is not maximally out or maximally up, it should be allowed to move to the maximally out position. When the table has returned to its maximally out position, normal operation can be resumed.

To model these requirements the event "*normal*" is introduced. Initially the system is in the state RESTRICTED (Fig. 4.7a). The table is only allowed to move out by means of the motor or by means of releasing the clutch. After the event *normal*, the system enters the state NORMAL: all movement events and clutch events are allowed. After occurrence of the event *TTR_on*, the system is in the state TTRON, and the motors are stopped. When TTR is turned off (*TTR_off*), the system enters the restricted state again.

The system can switch over to normal operation if it can be ensured that the system stays either maximally out or maximally in (Fig. 4.7b). Normal mode is represented by the states V̂HN, VHN and VĤN. In these 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 *TTR_on*, any horizontal or vertical position can be reached (corresponding to the states V̂HN̂, VHN̂, VĤN̂ and V̂ĤN̂. Notice that in Fig. 4.7b there is no state V̂ĤN. Because this state can in principle be reached by *uncontrollable* events, controller synthesis disables *controllable* events to ensure that the state is not reached. Therefore, in normal mode, when the table is not maximally up, the clutch must be enabled and horizontal movement is prohibited. Furthermore, vertical movement is prohibited if the tabletop is not maximally out (or not present).

5 User input

The user can control the system by means of a manual button and a tumble switch. The manual button switches the table to manual mode, and back. The tumble switch determines the motorized movement when manual mode is off.

5.1 Plant model (*UserI*)

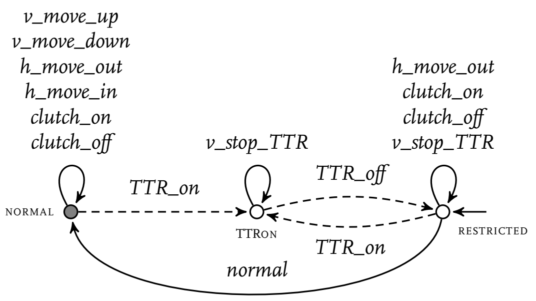
The manual button first emits the event *manual-pushed*. Subsequently, after a time delay, it emits the event *manual-timeout*. The manual button can be pushed infinitely often before the timeout occurs. For every push a new timeout is generated. Modeling this behavior would result in an infinite model. Therefore, this button is modeled with one location in which the two events are looped (Fig. 4.8a).

The tumble switch can either be up, down, or neutral, and emits events accordingly (Fig. 4.8b). The switch always returns to the neutral state, by physical construction.

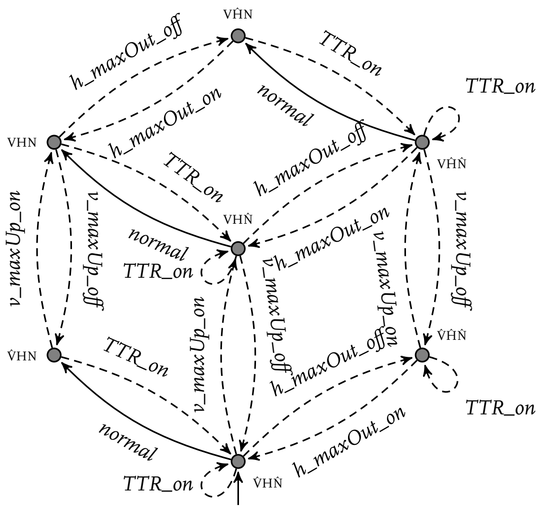
5.2 Control requirements (*Ureq*)

Pushing the manual button toggles the state of the clutch, if allowed. It may for instance not be allowed when moving horizontally. If it is not allowed to toggle the clutch, a timeout occurs after which the manual push event is ignored (Fig. 4.9a).

The position of the tumble switch determines the movement of the table. When the tumble switch is up, the table may move up and into the bore. When the switch is in the downward

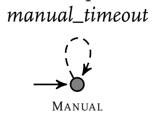


(a) hv_modes

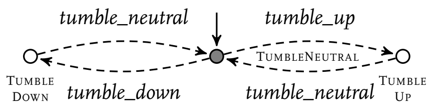


(b) hv_safe

Figure 4.7: Model *HVreq*: horizontal and vertical restrictions

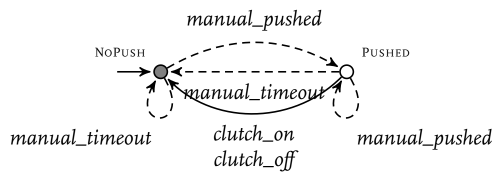


(a) *manual_button*

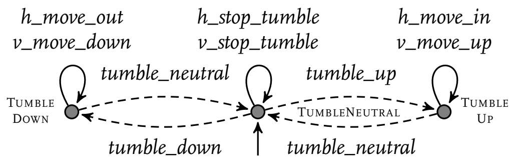


(b) *tumble_switch*

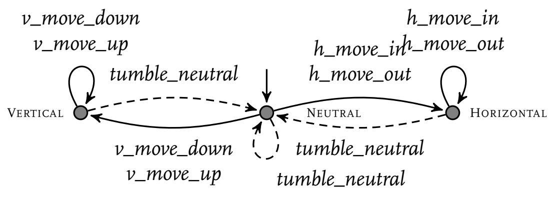
Figure 4.8: Model *UserI*: user input



(a) *manual_clutch*



(b) *tumble_move*



(c) *hv_switch*

Figure 4.9: Model *Tumblereq*: tumble switch requirements

position, the table is allowed to move out of the bore and down. When the tumble switch is in its neutral position, all movement should be stopped (Fig. 4.9b). The tumble switch must return to neutral before movement along the other axis is allowed (Fig. 4.9c).

6 Synthesis

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

$$Vaxis \parallel Haxis \parallel UserI$$

This model consists of 672 states and 14.384 transitions.

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

$$Vreq \parallel Hreq \parallel HVreq \parallel Ureq$$

This specification consists of 4.128 states and 34.884 transitions.

From these two models, the supervisor is generated using the LTCT tool [2]. The resulting supervisor contains 2.976 states and 22.560 transitions.

7 (Hardware-in-the-loop) simulation

The supervisor satisfies the control requirements by construction. The plant model or control requirements could however contain errors. To check for these errors, the synthesized supervisor is validated by means of simulation and thereafter the supervisor is tested on the real system by means of hardware-in-the-loop simulation.

7.1 Simulation

To validate the synthesized supervisor by means of simulation, the uncontrolled plant is modeled using the Compositional Interchange Formalism results in a hybrid (combined discrete-event/continuous time) model for the uncontrolled plant. To simulate the plant under supervision of the supervisor, we have defined and implemented a translation that takes as input the supervisor that we obtained from the LTCT tool and returns as output an equivalent CIF automaton. The parallel composition of the hybrid plant model and the CIF model of the supervisor has been simulated.

7.2 Hardware-in-the-loop simulation

The sensors and actuators of the actual patient support table are connected to an industrial grade I/O controller, which in turn is connected to a standard PC. The I/O controller conditions the sensor signals, translates sensor state changes to events, and translates events from the PC to appropriate inputs for the actuators. On the PC, the events from the I/O controller are buffered in an event queue. After receiving an event from the I/O controller, the state of the supervisor is updated, and the set of controllable events that is allowed by the supervisor is calculated. From this set, an event is selected and sent to the I/O controller.

In the simulation model, the plant model and the model of the supervisor interact synchronously, i.e. they synchronize on common events. However, during the hardware-in-the-loop simulation, the interaction between the patient table and the supervisor is asynchronously. More precisely: after a change of state of a sensor, this change has to be detected by the I/O controller (sensor polling delay). Then the I/O controller sends an event to the PC (communication delay between I/O controller and PC). After detection of the event (event queue polling delay), the state of the supervisor is updated, the allowed events are calculated, and an event is selected (calculation delay). In the setup, first all events from the event queue are processed. Then, when the event queue is empty, an allowed controllable event is selected and sent to the I/O controller.

8 Evolution

During a demonstration of the hardware-in-the-loop simulation, different behavior of the table was requested. Using the current control systems (the deployed ones as well as the generated supervisor from this paper), 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, which was considered to be unintuitive for the user. The desired new behavior in this case was that when the tumble switch is switched up, the table should first move out, and when the table has reached the maximally out position, the table should move up (iff the tumble switch is still up).

To model this requirement, the h_move_out is replaced: $h\_move\_out \triangleq h\_move\_out\_nor$, *h_move_out_rest*. In Fig. 4.5a, 4.6a,(b),(c),(e) and 4.9c all occurrences of event *h_move_out* are replaced. In Fig. 4.7a, the loop for event *h_move_out* at the state NORMAL is replaced by *h_move_out_nor*, and in state RESTRICTED the loop is replaced by *h_move_out_rest*. Finally, in Fig. 4.9b the event *h_move_out* is replaced by *h_move_out_nor*, and the event *h_move_out_rest* is added to the loops of states TUMBLEDOWN and TUMBLEUP.

This new requirement has been implemented on the actual patient support table four hours after it was conceived.

Chapter 5
Conclusions

In this report, supervisory control theory has been used to synthesize a supervisory controller for a patient support system of an MRI scanner. The uncontrolled system as well as the control requirements are modeled independently by means of small, loosely coupled automata. As a result, changes in the plant or in the control requirements can be realized quickly, without introducing errors. The improved evolvability of this approach has been demonstrated by operating the synthesized controller on the actual table, and by realizing a user request for improved functionality within few hours.

Bibliography

[1] N.E. Fuchs. Specifications are (preferably) executable. *Software Engineering Journal*, 7(5):323–334, 1992.

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