Supervisory control synthesis for a patient support system

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

Abstract— **Supervisory control theory (SCT) provides a formal approach to supervisory controller synthesis. In this paper, SCT 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 and minimally 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 table.**

I. Introduction

Present day complex systems are usually not designed from scratch. They have evolved during multiple generations of the system. Due to market demands and increasing competition, the number of features, and thus the complexity of systems increases, while the time-to-market of a system should be decreased. At the same time, systems must meet high quality constraints. This necessitates the need for methods to maximize reuse and to minimize the effort to design new generations of a system.

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.

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

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

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.

To support the evolvability of systems, the design process of supervisory controllers can be automated. By means of supervisory control theory (SCT), initiated by Ramadge/Wonham [1], 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.

In this paper, SCT is applied to a patient support system. This system is used to position patients in an MRI scanner, as shown in Fig. 1. 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.

The remainder of this paper is as follows: Section II discusses related work. Section III introduces the patient support case. Sections IV–VIII present the models of the patient support system and the control requirements. Section IX describes the synthesis of the supervisor. Section X presents the validation of the supervisor by simulation, and Section XII presents concluding remarks.

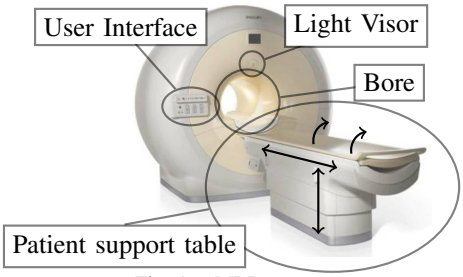


Fig. 1. MRI scanner

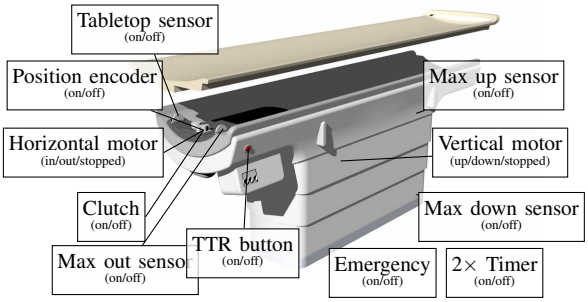


Fig. 2. Patient table

II. Related work

Despite the fact that SCT is well established, industrial application is rare. The main causes of this can be found in the computational complexity, model interpretation and modeling and implementation effort [2], [3].

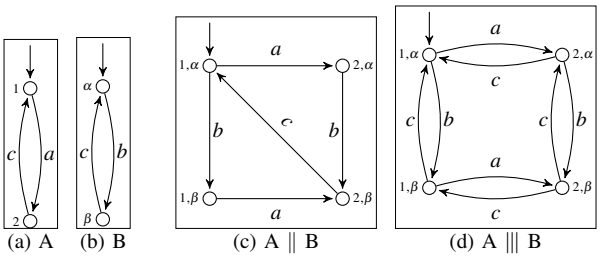
Previous applications of SCT include:

- A rapid thermal multiprocessor [2], to allow flexible design and reliable update of processing "recipes".
- Mobile robots [3], [4], to perform predefined tasks, while preventing collisions.
- A water bath boiler [5], where the boiler is operated to maintain a continuous steam supply.
- Under-load tap-changing transformers [6], to control voltage in electrical power systems.
- Automated manufacturing and assembly systems, see [7], [8], [9], [10], [11], among others. In these systems, different workstations are connected by transport equipment. The main task is to manufacture/assemble products according to some recipes.

The contribution of this paper is threefold. First, it deals with a true high-tech industrial application (no mock-up). Second, the specification consists of small, loosely coupled automata to improve evolvability. Third, the implementation is based on the Compositional Interchange Format (see Section X), which ensures interoperability of the implementation with a large set of languages and tools.

III. Patient support case: introduction

The patient support system is used to position a patient in an MRI scanner (Fig. 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 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).



(a) A (b) B (c) A ‖ B (d) A ||| B

Fig. 3. Parallel and interleaving composition

The patient support system (Fig. 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), 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.

IV. Notations and model semantics

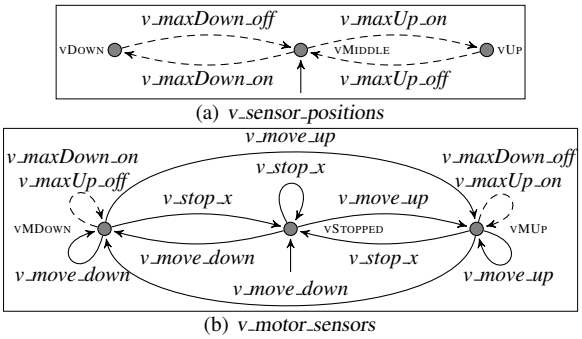
In the models, states are denoted by vertices, initial states are indicated by an unconnected incoming arrow, and marked states are denoted by filled vertices. Controllable and uncontrollable events are drawn with solid and dashed edges, respectively. Multiple events on a edge represent an edge for each 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. Fig. 3 shows an example of each of the two parallel composition operators.

V. 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.

A. Plant model (VerticalAxis)

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



(a) *v_sensor_positions*

(b) *v_motor_sensors*

Fig. 4. Model *VerticalAxis*: vertical axis

v_max... _on or *v_max... _off*, when a sensor becomes active or ceases to be active, respectively (Fig. 4a). 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. 4b). 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.

B. Control requirements (VertRequire)

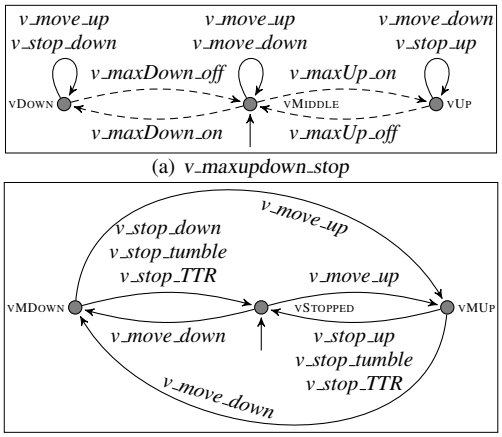
In Fig 4b, the event *v_stop_x* denotes a number of different of stop events, namely *v_stop_x* $\triangleq$ *v_stop_up*, *v_stop_down*, *v_stop_TTR*, *v_stop_tumble*. This allows to model the stop behavior independently. The stop event should be disabled most of the time, however it should be enabled 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 event *v_move_x* is an abbreviation for the move events, *v_move_x* $\triangleq$ *v_move_up*, *v_move_down*.

The vertical axis should not move beyond its maximally up and maximally down position (Fig. 5a). 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. These repetitions can result in an infinite string of the same controllable events, without resulting in meaningful behavior of the system. Therefore, repetitions are disabled in the control requirements. The requirement in Fig. 5b 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.

VI. Horizontal axis

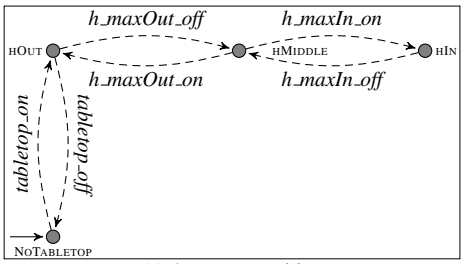
The horizontal axis contains four sensors: maximally in, maximally out, tabletop present and tabletop release active; and two actuators: the horizontal motor and the clutch. The tabletop can only be added and removed in the maximally out position. The clutch connects the motor to the tabletop.



(a) *v_maxupdown_stop*

(b) *vmotor_sequence*

Fig. 5. Model *VertRequire*: vertical control requirements



(a) *h_sensor_position*

Fig. 6. Model *HaxisA*: horizontal axis synchronizing

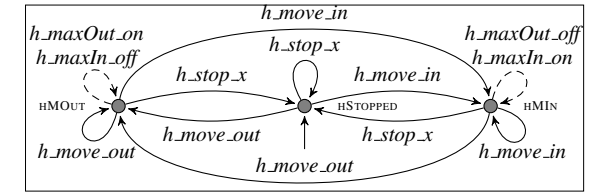
When the clutch is applied, the motor controls the movement of the tabletop. Otherwise the tabletop can be moved freely by the operator. The tabletop release (TTR) is a hardware safety system which releases the clutch independently of the controller. If TTR is active, the table can be moved freely, even if the controller enables the clutch.

A. Plant model (HorizontalAxis)

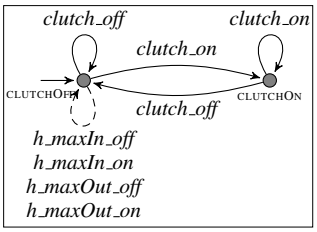
The model *HorizontalAxis* of the horizontal axis is divided into two parts: *HorizontalAxis* $\triangleq$ *HaxisA* $\parallel$ *HaxisB*. The automata in the part *HaxisA* are composed by normal (synchronizing) parallel composition. The automata in the part *HaxisB* are composed by interleaving parallel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

Fig. 6 (*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.

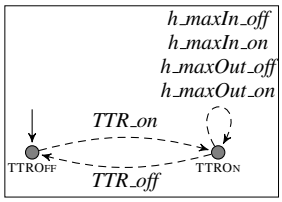
Fig. 7 (*HaxisB*) models the following behavior: only when the tabletop is moving, the maximally in and out sensors can switch. The tabletop moves when it is moved by the motor drive (Fig. 7a, which is analogous to Fig. 4b). It can also be moved by an operator when the clutch is released (Fig. 7b) or when TTR is active (Fig. 7c). Because the sensors can switch in either case, these three models are composed by interleaving parallel composition, i.e. *HaxisB* $\triangleq$ *h_motor_sensors* $|||$ *clutch_sensors* $|||$ *TTR_sensors*.



(a) *h_motor_sensors*



(b) *clutch_sensors*



(c) *TTR_sensors*

Fig. 7. Model *HaxisB*: horizontal axis interleaving

B. Control requirements (HorRequire)

Analogous to the event *v_stop_x*, the event *h_stop_x* denotes a number of different stop events, namely *h_stop_x* $\triangleq$ *h_stop_in*, *h_stop_out*, *h_stop_TTR*, *h_stop_tumble*. The event *clutch_x* is an abbreviation for the clutch events, *clutch_x* $\triangleq$ *clutch_on*, *clutch_off*, the event *h_move_x* is an abbreviation for the move events, *h_move_x* $\triangleq$ *h_move_in*, *h_move_out*.

Similarly to the vertical axis (Fig. 5a), the horizontal axis may not move beyond its maximally in and out position (Fig. 8a). 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. 8b), 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. 8c), 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. Furthermore, only if TTR is turned off, the clutch may be turned on or off.

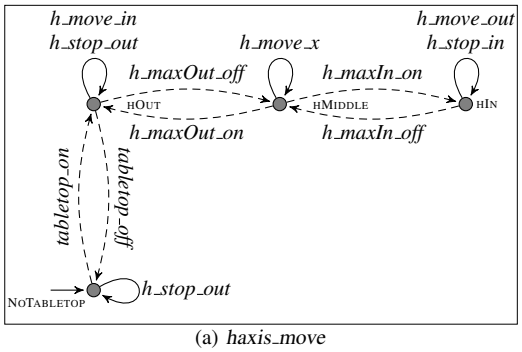
Fig. 8d disables repetitions of clutch events, and in analogy to the vertical axis (Fig. 5b), Fig. 8e prevents repetitions of motor events.

VII. Horizontal and vertical axis

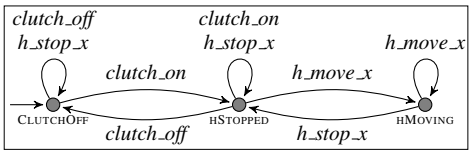
There is no physical interaction between the transducers of the horizontal and vertical axis. Therefore, no new plant models need to be introduced. Only control requirements need to be added.

A. Control requirements (HorVertRequire)

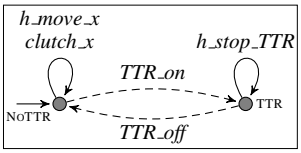
Collisions of the tabletop with the magnet should be prevented. To accomplish this, the tabletop must be either maximally out (or not present), or the table must be maximally up. However, this condition can be violated when TTR is active. The table can then be moved freely by the operator, which may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



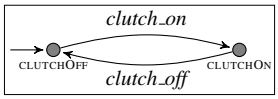
(a) *haxis_move*



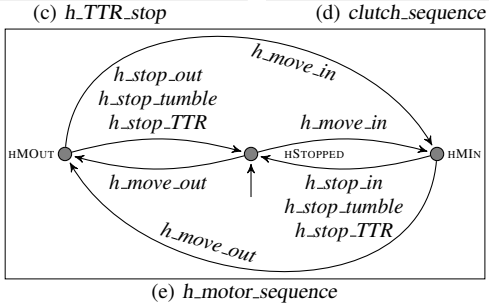
(b) *h_clutch_move*



(c) *h_TTR_stop*



(d) *clutch_sequence*

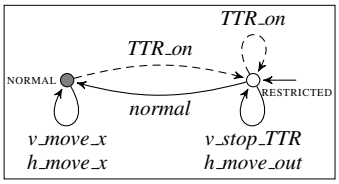


(e) *h_motor_sequence*

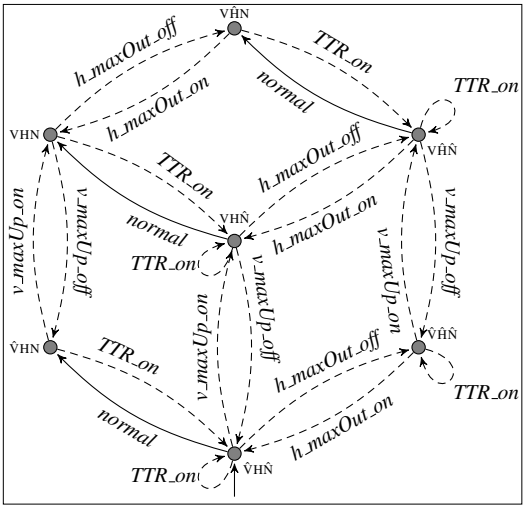
Fig. 8. Model *HorRequire*: horizontal control requirements

To model these requirements the event "*normal*" is introduced. Initially the system is in the state RESTRICTED (Fig. 9a). The table is only allowed to move out by means of the motor, vertical movement is stopped. Note that horizontal movements are stopped due to control requirements for the horizontal axis. After the event *normal*, the system enters the state NORMAL, all movement events are allowed. After occurrence of the event *TTR_on*, 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. 9b). Normal mode is represented by the states $\hat{\text{V}}$HN, VHN and V$\hat{\text{H}}$N. 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. $\hat{\text{V}}$ represents not vertically maximally up. After an event *TTR_on*, any horizontal or vertical position can be reached (corresponding to the states $\hat{\text{V}}$H$\hat{\text{N}}$, VH$\hat{\text{N}}$, V$\hat{\text{H}}\hat{\text{N}}$ and $\hat{\text{V}}\hat{\text{H}}\hat{\text{N}}$. Notice that in Fig. 9b there is no state $\hat{\text{V}}\hat{\text{H}}$N. 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

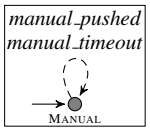


(a) hv_modes

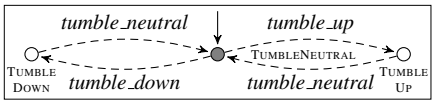


(b) hv_safe

Fig. 9. Model *HorVertRequire*: horizontal and vertical restrictions



(a) *manual_button*



(b) *tumble_switch*

Fig. 10. Model *UserInterface*: user input

movement is prohibited if the tabletop is not maximally out (or not present).

VIII. 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.

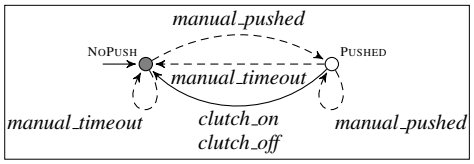
A. Plant model (UserInterface)

The manual button first emits the event *manual-pushed*. Subsequently, after a time delay, it emits the event *manual-timeout*. In theory, 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. 10a).

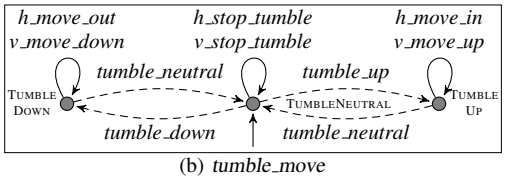
The tumble switch can either be up, down, or neutral, and emits events accordingly (Fig. 10b). The switch always returns to the neutral state, by physical construction.

B. Control requirements (UserRequire)

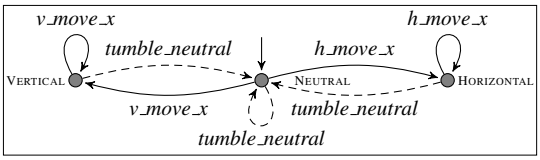
Pushing the manual button toggles the state of the clutch, if allowed. It may for instance not be allowed when moving horizontally. If it is not allowed to toggle the clutch, a timeout occurs after which the manual push event is ignored (Fig. 11a).



(a) *manual_clutch*



(b) *tumble_move*



(c) *hv_switch*

Fig. 11. Model *UserRequire*: tumble switch requirements

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 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. 11b). The tumble switch must return to neutral before movement along the other axis is allowed (Fig. 11c).

IX. 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:

$$\mathit{VerticalAxis} \parallel \mathit{HorizontalAxis} \parallel \mathit{UserInterface}$$

This model consists of 672 states and 14.384 transitions.

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

$$\mathit{VertRequire} \parallel \mathit{HorRequire} \parallel \mathit{HorVertRequire} \parallel \mathit{UserRequire}$$

This specification consists of 4.128 states and 34.884 transitions.

From these two models, the supervisor is generated using the LTCT tool [1]. Calculation takes less then a second on an average computer. The resulting supervisor contains 2.976 states and 22.560 transitions.

X. (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.

A. Simulation

To validate the synthesized supervisor by means of simulation, the uncontrolled plant is modeled using the Compositional Interchange Formalism (CIF) [12], [13]. This 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.

B. 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.

XI. 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. 7a, 8a,b,c,e and 11c all occurrences of event *h_move_out* are replaced. In Fig. 9a, 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. 11b 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.

XII. Concluding remarks

In this paper, 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.

The system consists of two disjoint axis, therefore it should be possible to synthesize controllers for each axis separately. However, a third supervisor is needed to control the interaction between the two axis. In this modular setting global non-blocking cannot be guaranteed. Defining modular supervisors for this system is future work.

References

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

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

[3] J. Liu and H. Darabi, "Ramadge-Wonham supervisory control of mobile robots: lessons from practice," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 1, 2002, pp. 670–675 vol.1.

[4] 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*, 1994, pp. 2557–2562 vol.3.

[5] M. Moniruzzaman and P. Gohari, "Implementing supervisory control maps with PLC," in *American Control Conference, 2007. ACC '07*, 2007, pp. 3594–3599.

[6] 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*, 2007, pp. 901–906.

[7] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," *Proc. Rensselaers 1994 Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, p. 319324, 1994.

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

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

[10] M. de Queiroz and J. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, 2002, pp. 377–382.

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

[12] D. A. v. Beek, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Foundations of an interchange format for hybrid systems," in *Hybrid Systems: Computation and Control, 10th International Workshop*, ser. LNCS, A. Bemporad, A. Bicchi, and G. Butazzo, Eds., vol. 4416. Pisa: springer, 2007, pp. 587–600.

[13] D. A. v. Beek, M. 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*, Seoul, Korea, 2008.