Chapter 2
Integration of Supervisory Control Synthesis in Model-Based Systems Engineering

Jos C.M. Baeten, Joanna M. van de Mortel-Fronczak and Jacobus E. Rooda

Abstract Increasing system complexity, time to market and development costs reduction place higher demands on engineering processes. Formal models play an important role here because they enable the use of various model-based analyses and early integration techniques and tools. Engineering processes based on formal models are able to cope with complexity. They also support time to market and development costs reduction. Moreover, application of supervisory control synthesis in the development of control systems can speed up the process considerably. This paper discusses the integration of recently developed supervisor synthesis techniques and tools in engineering processes. To illustrate this approach, examples of industrial cases are presented, where supervisors synthesized have successfully been implemented and integrated in existing resource control platforms.

2.1 Introduction

In current industrial practice, it is very difficult to deal with high-tech multidisciplinary system development due to system complexity, market pressure, and resource limitations. To overcome the difficulties, various kinds of models are used increasingly often in the development process. Specifically, formal and executable models built and employed in the design phase can be used to assess functional correctness and performance of component designs and overall system design. Formal verification, in particular model checking [3], is employed when a high degree of confidence in functional correctness of a design is required. To assess design performance, one

J.C.M. Baeten
Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
e-mail: jos.baeten@cwi.nl

J.C.M. Baeten · J.M. van de Mortel-Fronczak (✉) · J.E. Rooda
Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: j.m.v.d.mortel@tue.nl

J.E. Rooda
e-mail: j.e.rooda@tue.nl

G.M. Dimirovski (ed.), *Complex Systems*, Studies in Systems, Decision and Control 55, DOI 10.1007/978-3-319-28860-4_2

can use simulation directly on the model, or analyze an associated Markov chain or queueing model.

Successful use of models is, however, not limited to the design phase. Models are nowadays also used in the test and integration phases. The effort required to integrate components and to test the resulting high-tech multidisciplinary system against the initial requirements increases significantly with the increase in complexity. The integration and test phases have a large and growing influence on time to market and product quality, being the main business drivers for system developers. High-tech multidisciplinary systems are systems in which cutting edge technologies from multiple engineering disciplines are integrated to meet strict quality requirements set by the customer. A characteristic of such systems is that they are integration intensive, which means that integrating components and testing the resulting system against its requirements consumes a large portion of total system development effort. In some cases, the integration and test effort may even exceed the already considerable amount of development effort required to deal with the complexity of components. Examples of high-tech multidisciplinary systems are commercial systems such as wafer scanners, electron microscopes, and high-speed printers; medical systems such as magnetic resonance imaging (MRI) scanners; as well as aerospace systems such as satellites, airplanes, and spacecraft.

To wit, it is shown in [6, 7] that models can successfully be used for early integration of wafer scanners, where the idea is to start integrating and testing as soon as possible, using models as replacements of components as long as actual component realizations are not yet available. Using an appropriate integration infrastructure that implements the designed and modeled interaction behavior, realized components can be integrated with models of not yet realized components. The resulting model-based integrated system can then be tested on system level before complete system realization is available. Another application is in model-based testing [8], where a model is used to automatically derive and execute tests on an implementation, as shown for wafer scanners in [24].

The use of models enables application of model-based techniques and tools for thorough system analysis and systematic testing, helping to improve system overview for engineers. Additionally, models of components and requirements enable synthesis of supervisory controllers essential for correct functioning of systems. Supervisor synthesis procedures are based on supervisory control theory originating in [26].

Figure 2.1 schematically illustrates a high-tech system with the focus on control. At the bottom, the main structure is depicted, containing the physical hardware components. Sensors and actuators are mounted on these hardware components to monitor their position or state and to actuate them. The resource control layer assures that actuators reach a desired position in a desired way (feedback control). The supervisory control layer coordinates individual components and gives the desired functionality to the system.

The high-tech industry is also confronted with problems related to systematic upgrades of complex control software, necessary either for new systems or machines or for extending functionality of the existing ones. Also, in this context, system component and requirement models can be used to speed up the supervisory control

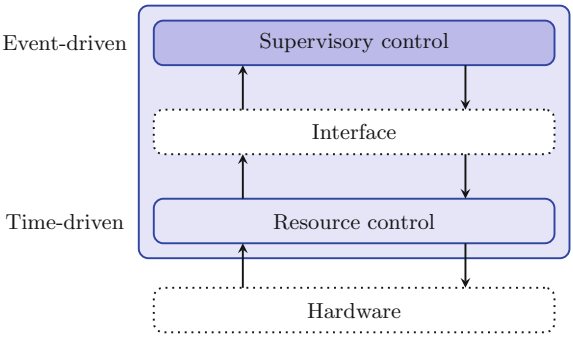

Fig. 2.1 Positioning supervisory control

design process. To embed supervisor synthesis in model-based engineering, distributed synthesis techniques are needed to handle the complexity frequently encountered in high-tech systems and to support the evolvability in the development process. High-tech companies are often challenged to increase the functionality and quality of a product, while simultaneously time to market and product costs should be reduced. Current practice shows that this is not straightforward. As a result, there is a need for new engineering processes. The purpose of this paper is to show how supervisory control theory introduced by Ramadge and Wonham [26] can contribute to system development and how it can be integrated in an engineering process. To this end, the applicability of recently developed supervisor synthesis techniques to industrial cases is discussed.

The paper is structured as follows. Section 2.2 gives a short overview of engineering processes that are based on models. In Sect. 2.3, we informally explain the basic supervisor synthesis procedure. Section 2.4 focusses on the role of model transformations in support of model-based engineering processes, especially for industrial applications. An overview of a few industrial cases in which a synthesis-based engineering method is applied is presented in Sect. 2.5. Section 2.6 concludes the paper by showing what impact supervisory control synthesis has on the system development process.

2.2 Overview

To support structured system development, several engineering processes are introduced originating from *systems architecting* [27] and *systems engineering* [19]. More recently, [20] proposed a conceptual model specifically supporting embedded systems architecting. In general, a system engineering process starts with a global definition of the requirements that the system should fulfill. Based on these requirements, the global system design is defined, usually partitioned into subsystems or modules. For each module, the requirements are derived from the global design, and a design is defined and then built. Every module is tested separately to assure that its requirements are satisfied. Subsequently, modules are integrated and the complete system

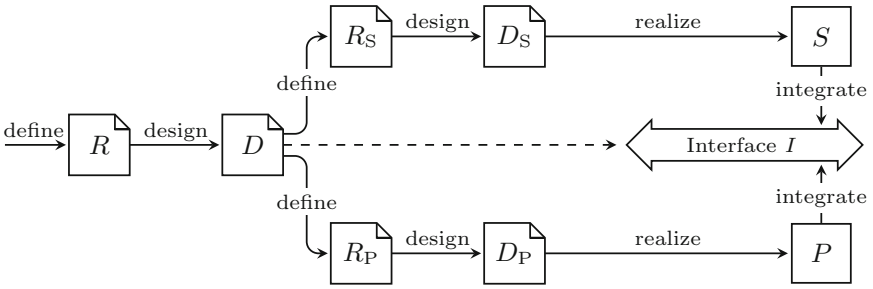

Fig. 2.2 Traditional engineering process

is also tested to assure that it complies to its requirements. For a system consisting of two modules, component *P* and its controller *S*, this is schematically depicted in Fig. 2.2.

Thanks to the introduction of microprocessors, it became possible to develop systems with substantially extended functionality. The extended functionality often implies a multidisciplinary character because mechanical and control engineers needed to cooperate with electrical, electronic, and computer science engineers. Especially for this kind of systems, traditional engineering processes are often not optimal in terms of system quality, time to market, and costs.

In the last decades, engineers started to use executable software models to test the designs before they are actually built. An advantage of such models is that they can be used not only to analyze designed system behavior but also to investigate how components or modules that are already built interact with the rest of the system that is not yet built. One can think of, for instance, checking if the system is nonblocking or estimating system performance. In [6], evidence is provided that executable models help in improving system quality and in decreasing time to market. Models also support evolvability, as they can easily be reused for similar systems. Integration of executable models in engineering processes is rendered in the scheme of Fig. 2.3.

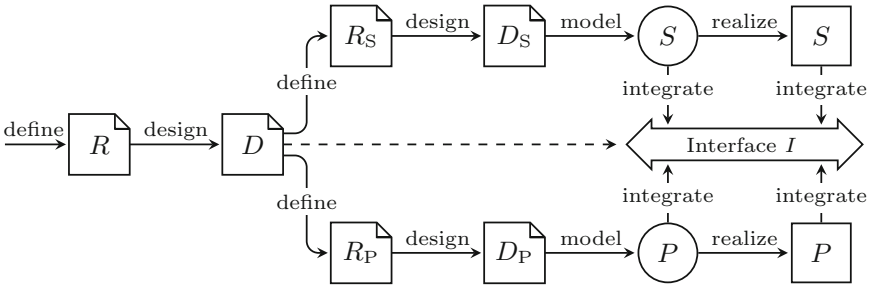

Fig. 2.3 Model-based engineering process

Certainly for complex systems, model-based systems engineering (MBSE) methods are state-of-the-art. However, all current MBSE approaches still carry a significant contribution to the overall system cost. The industry faces increasing complexity and is challenged to reduce development time and cost. To meet these challenges, a next-generation MBSE approach is required that reduces human errors and shortens the development cycle. An overview of the state-of-the-art commercially available MBSE methodologies, such as IBM Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD), and Vitech MBSE Methodology, can be found in [10]. They include processes, methods, and tools supporting manual development of systems, therefore helping in achieving a paradigm shift from traditional document-based approach to model-based approach. However, in the context of control software design, they focus on design model *formulation*. To cope with industrial challenges, more powerful techniques and tools are required. The MBSE process proposed in [7] enables the use of various analysis techniques and tools based on formal models to support system development. Significant development cost and cycle time reduction can be gained by the incorporation of methods and tools for supervisor synthesis. To this end, as opposed to traditional engineering, an explicit separation of concerns related to the plant and the supervisor is necessary. This also means that requirements for supervisory control must be separated from requirements for regulative control. Supervisor synthesis techniques replace manual design of control logic by automatical *derivation* from component and requirement specifications. The synthesized control logic is nonblocking and by construction correct with respect to safety properties. In general, liveness properties still need to be verified. The position of supervisor synthesis in the development process is shown in Fig. 2.4 [28].

This setting partly corresponds to Fig. 2.3. As previously, requirements R of the system under supervision are defined first. Based on these requirements, design D of the system and a decomposition into the uncontrolled plant and the supervisor are defined. After decomposition, requirements R_S for the supervisor and R_P for the uncontrolled plant are specified. From the plant requirements, design D_P and one or more plant models P can be defined. However, instead of making a design and a model of the supervisor, the requirements for the supervisor are formally modeled.

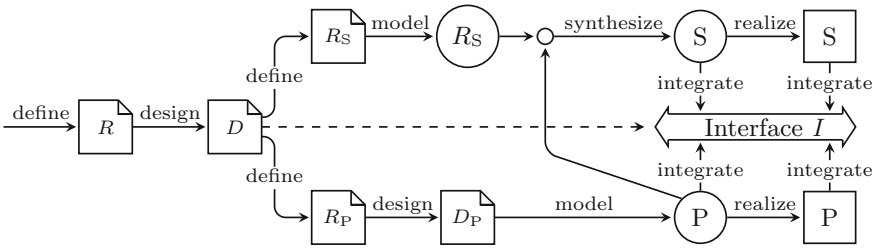

Fig. 2.4 Engineering process with supervisory control synthesis

A discrete-event model of the plant and the formal model of R_S can be used to *synthesize* a supervisor in the framework of supervisory control theory. Plant models can also be used to analyze the behavior of the uncontrolled plant under supervision of the supervisor. Formal and executable models built and employed in the design phase can be used to assess functional correctness and performance of the design of components and the overall system. One example of commercially available tool used for correctness checking is the Communicating Sequential Processes (CSP)-based tool of Verum (http://www.verum.com). To assess performance of the design, one can use simulation directly on the model, or use analytical or numerical methods to analyze an associated stochastic model.

In synthesis-based engineering, required system properties are used as input for generation of a design of a supervisor that is correct by construction. As a consequence, in the analysis phase verification can be eliminated to a large extent. This changes the development process from implementing and debugging the design and the implementation, to defining the requirements.

2.3 Synthesis-Based Supervisory Control Engineering

Supervisory control theory (SCT) of [26] allows to synthesize a model of the supervisor from formal models of the uncontrolled system and of the requirements. First, the uncontrolled system (plant P) is formally specified in terms of automata. A plant automaton describes the physically possible behavior of the system to be controlled. Then, the requirement specifications (R_S) for the system under control are formally defined in terms of automata. A model of the supervisor (S) is generated from these formal models.

The resulting supervisor can be used to supervise an uncontrolled plant, which is schematically illustrated in Fig. 2.5. The supervisor can only react to observable events that are generated by the uncontrolled plant. The supervisor influences the behavior of the plant by disabling certain controllable events. The method guarantees that the system consisting of the derived supervisor and the uncontrolled plant fulfills the requirements. If the supervised model does not contain the desired functionality, the models of the uncontrolled plant or the requirements are inadequate.

Supervisory control problem is formulated as follows. For plant P and requirement R, a supervisor S is required such that:

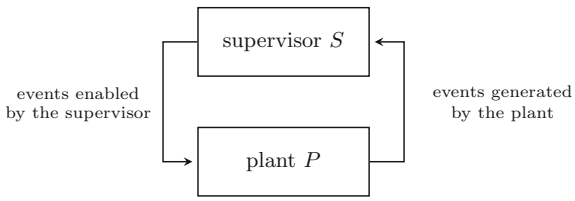

Fig. 2.5 The feedback loop of supervisory control

1. P under control of S, denoted by S/P, satisfies requirement R.
2. S does not disable uncontrollable events.
3. Output of S only depends on observable outputs of P.
4. S/P is nonblocking.
5. S is optimal, that is, maximally permissive.

Supervisory control theory provides means to synthesize S. It is a conceptually simple framework based on discrete-event system models. Different methods have been developed that allow for an automatic synthesis of a supervisor. As computational complexity is high for systems of industrial size, several advanced techniques have been introduced to reduce synthesis complexity, such as modular [25], hierarchical [35], interface-based hierarchical [14], aggregative [11, 17], distributed [9, 13, 32], coordinated distributed [29], and aggregated distributed [30]. They allow for splitting a system in a number of modules for which local supervisors can be synthesized. If needed, also a coordinator can be synthesized on top of them. The local supervisors together with the coordinator compose a nonblocking supervisor for the total system. A different approach is the state-based supervisory control framework of [16]. In this framework, discrete-event systems are represented by state tree structures (STS), a representation that allows for computationally efficient monolithic supervisor synthesis. Unlike the event-based framework, the state-based framework allows to formulate requirements as conditions over states. In [2], a first step is reported in the direction of defining supervisor synthesis techniques for a process-theoretic setting. Additionally, [31] describes how to compute a nonblocking supervisor that not only complies with the prescribed requirements but also achieves a time optimal performance such as throughput.

In the context of supervisory control, system component models are defined by automata. Often, each component and its resource controller are modeled together by one automaton. Automata consist of relevant states of each resource and transitions labeled by controllable and uncontrollable events. Controllable events represent relevant discrete commands to the resource control that can be enabled or disabled by the supervisor. Uncontrollable events represent messages from the resource control to the supervisor that cannot be disabled by the supervisor. The conditions the supervisor must satisfy are formally defined by expressions on languages generated by the automata.

To illustrate the basic supervisor synthesis procedure, a simple example from [36] is worked out below. A workcell consists of two machines M_1 and M_2, and an automated guided vehicle (AGV). The AGV can load a workpiece at M_1 or M_2 and unload it at M_2 or at the infinite buffer B, respectively. This is schematically shown in Fig. 2.6.

In this paper, we use italic event labels (e.g., c) and bold state labels (e.g., **Busy**). In the graphical automaton representation, initial states are denoted by an unconnected incoming arrow and marker states are denoted by an unconnected outgoing arrow. Controllable and uncontrollable events are denoted by solid and dashed edges, respectively.

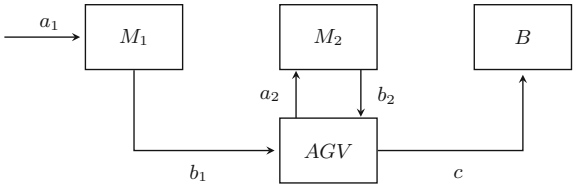

Fig. 2.6 Workcell

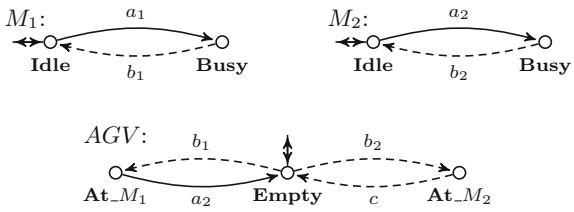

Fig. 2.7 Workcell automata

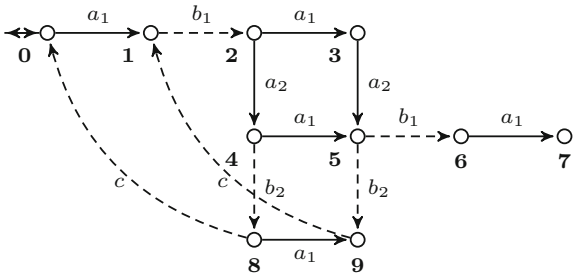

Fig. 2.8 Uncontrolled system

Components M_1, M_2, and AGV are modeled by automata shown in Fig. 2.7. The behavior of the uncontrolled system P is represented by the synchronous product of M_1, M_2, and AGV drawn in Fig. 2.8.

We informally explain the conditions that must be satisfied by the supervisor for this uncontrolled system. For simplicity, we assume the requirement does not impose any restrictions on admissible system behavior and all events are observable. This implies that conditions 1 and 3 are satisfied. Hence, we can focus on the remaining conditions. In the basic supervisor synthesis approach, the supervisor is derived from the intersection of the two languages: the uncontrolled plant behavior and the admissible behavior defined by the requirement. Because of the assumption, in our case it suffices to take the first one into account. Hence, we must check whether the plant is nonblocking. As can be seen above, the absence of control results in a blocking situation (deadlock in state 7). The supervisor we are constructing should influence the plant under control to avoid the blocking situation. Only controllable events may be disabled. To avoid blocking, controllable event a_1 must be disabled in state 4 and controllable event a_2 must be disabled in state 3. This results in Fig. 2.9a, where these supervisory control actions introduce a new blocking situation, state 3. This situation can be avoided by disabling controllable event a_1 in state 2 resulting

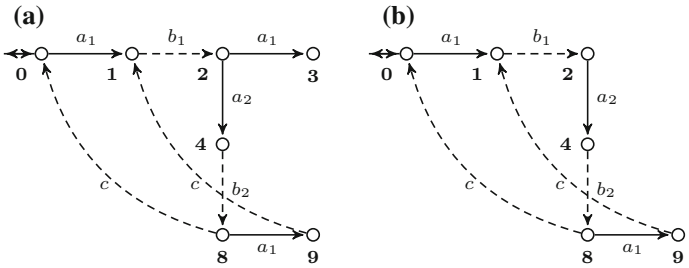

Fig. 2.9 Derivation of the supervisor. **a** Step 1. **b** Step 2

in Fig. 2.9b. Finally, by construction this supervisor gives a proper optimal control to the plant: conditions 1 through 5 are satisfied.

Supervisors can be synthesized according to the procedure sketched above. In a similar way, a self-made supervisor can be verified by checking the enumerated conditions.

In Sect. 2.5, a few synthesis techniques from the event-based and state-based frameworks are applied in industrial case studies.

2.4 Model Transformations

The main difference between the traditional engineering and the model-based engineering processes is the inclusion of models in the latter. System components are modeled before being realized. This extra step in the development process has several advantages, as mentioned in [7]:

- Abstract models give a systematic approach to specify component and system behavior with more consistency and less ambiguity than documents.
- Models make it easier to analyze dynamic behavior as well as the performance of a component of a system.
- Using various model-based analysis techniques indicated in Fig. 2.10, errors can be encountered in an early stage of the system development process when no component is realized, which decreases risk.

By simulating and validating models, the actual behavior can be analyzed in an early stage of the development process. However, simulation can only show that models might have correct behavior. It cannot guarantee model correctness. If models are validated and simulated and the conclusion is drawn that the system has the correct functionality, components can be realized. Formal verification techniques can be used to prove model properties.

Available components can be tested with hardware-in-the-loop simulation and testing. Composition of models and realizations can be tested as a whole to check for

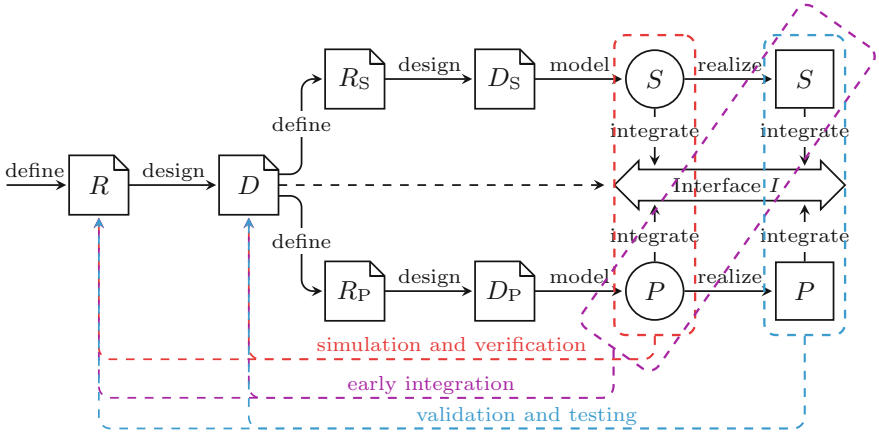

Fig. 2.10 Model-based analysis techniques

the correct functionality, which is called early integration. Also in this step, design errors can be encountered in an early stage, which decreases risk. If all components are realized, all models can be replaced by their realizations. The final implementation can be tested to verify if the system as a whole fulfills its requirement *R*. In this phase, model-based testing can be applied, where models are used to automatically derive and execute tests on the implementation.

In spite of the numerous advantages, model-based analysis techniques can only help the engineer with discovering errors in an early stage. There is no formal link between models and implementations. In other words, model correctness does not guarantee implementation correctness. Supervisory control theory explained in Sect. 2.3 provides this formal link and eliminates the manual design of the supervisor. This theory allows the generation of a supervisor that is mathematically correct with respect to formal plant models and requirements. As a consequence, design and implementation do not need to be tested against the requirements, the verification can be eliminated to a large extent, and the engineer can focus on system validation.

To support model-based systems engineering in a consistent way, model transformations are needed. The Chi language [4] has been developed to provide a process algebraic generic modeling formalism for hybrid systems. The language supports hierarchy and modularity to deal with large-scale systems, by providing operators for model reuse, parallelism, and nesting. It is based on processes that interact by shared variables, by communication via shared channels, and by synchronization by means of shared actions. Furthermore, arbitrary differential and algebraic equations are supported, for modeling of continuous time behavior. The Chi toolset provides translations to various verification tools, discrete events, and hybrid simulators, and supports hardware-in-the-loop simulation and control prototyping. Additionally, it facilitates code generation for various platforms.

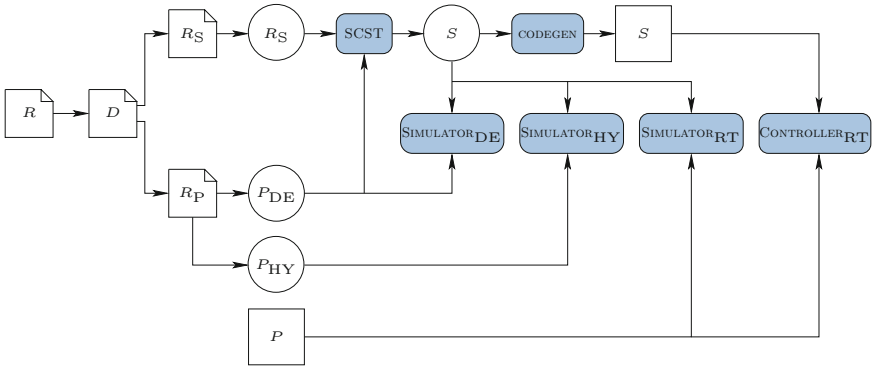

Fig. 2.11 Toolchain for model-based systems engineering

In Fig. 2.11, the Chi toolchain is depicted that supports model-based systems engineering with supervisor synthesis as described in [28]. In this figure, tools are depicted in blue, models are circles, and realizations are rectangles. The uncontrolled discrete-event behavior of the plant is formally modeled by automata represented by P_{DE}. Control requirements R_S are formally modeled by automata or logical expressions. From these models, a supervisor can be synthesized using a supervisor control synthesis tool (SCST) and translated to the Chi language, resulting in a model of the supervisor S. Three tools are used for supervisor synthesis and translations exist to and from the associated file formats:

- Translation Communication Tool (TCT) of [36] (http://www.control.utoronto.ca/cgi-bin/dldes.cgi) and Supervisor Synthesis Package (SSP) of Systems Engineering Group from Eindhoven University of Technology (http://se.wtb.tue.nl/sewiki/supcon/start) for the event-based supervisory control framework.
- NBC tool for the state-based supervisory control framework of [16].

The Chi simulator can be used to simulate the model of the supervisor S together with the discrete-event plant model P in order to analyze its behavior with respect to the control requirements. Furthermore, the discrete-event plant model can be replaced by the hybrid plant model P_{HY} or by real hardware in order to test whether it complies with the models. The supervisor model can also be used for code generation (CODEGEN). It can be implemented on a Personal Computer (PC) or Programmable-Logic-Controller (PLC) real-time control platform connected to the actual plant hardware.

In the next section, the model-based systems engineering approach supported by the toolchain described above is applied to three industrial cases in order to obtain a model of the supervisor, perform model-based system analysis, derive a supervisor implementation, and test it on the existing system.

2.5 Industrial Cases

Supervisory control theory is well established and received substantial attention in research papers. Although several applications reported in the literature, e.g., [15, 22, 23], include implementation of synthesized supervisors on a PC or a PLC, they are usually put in practice in an academic setup. To our knowledge, the industrial applications described in [12, 18, 34] are ones of the few that actually refer to an existing industrial product. Recently, a project is started in the automotive application area, where supervisor synthesis techniques are used in the design of the discrete part of a cruise control system.

2.5.1 MRI Scanner

In [34], supervisor synthesis is applied to a support system used to position patients in an MRI scanner of Philips Healthcare (http://www.medical.philips.com). MRI stands for magnetic resonance imaging, which is a medical imaging technique used in radiology to visualize detailed internal structures. An MRI scanner is used for noninvasive medical diagnosis. It uses a powerful magnetic field to align the magnetization of some atoms in the body, and nonionizing radio frequency fields to systematically alter the alignment of this magnetization. This causes the nuclei to produce a rotating magnetic field detectable by the scanner. This information is recorded to construct an image of the scanned area of the body. In clinical practice, MRI is used to distinguish pathologic tissue (such as a brain tumor) from normal tissue. One advantage of an MRI scan is that it is harmless to the patient, unlike Computed Tomography (CT) scans and traditional X-rays, which both use ionizing radiation. The traditional MRI scanner is a large cylinder-shaped tube surrounded by a circular magnet, called the bore. The patient lies on a moveable examination table, called the patient support system, which slides into the center of the magnet, see Fig. 2.12.

To properly control the patient support system, several actuators and sensors, as well as the extended user interface, must be taken into account. The modeled part

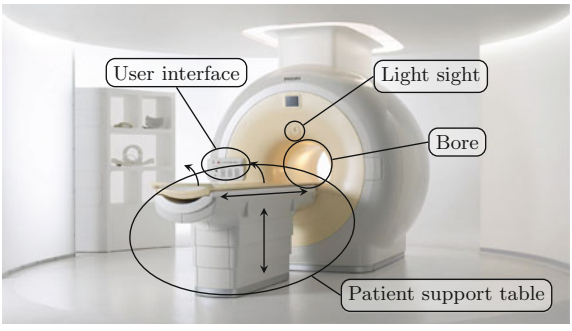

Fig. 2.12 The MRI scanner

of the system is divided into three modules: the vertical axis, the horizontal axis, and the user interface. The vertical axis consists of the lift driven by the vertical motor drive and associated end-position sensors. The horizontal axis contains the tabletop that can be moved into and out of the bore, either manually or by applying the horizontal motor drive depending on the state of the clutch. It also contains sensors detecting the presence and the position of the tabletop. The tabletop can only be placed or 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 that releases the clutch independently of the controller. If TTR is active, the table can be moved freely, even if the controller enables the clutch. The user interface allows the operator to use the manual button and the tumble switch. The manual button is used to switch between the manual and automatic mode of the tabletop movements. The tumble switch determines whether and how the patient support table should move. The patient support system model does not include the hardware safety systems, the light visor for marking, and automated positioning of the scan plane in the bore.

If no tabletop is present, horizontal movement is not allowed and the associated motor should be stopped. The tabletop may only be moved by the horizontal motor if the clutch is applied and the TTR is not active. 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. The 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. Only if TTR is turned off, the clutch may be turned on or off.

Control requirements are related to tabletop handling. Four safety-related aspects can be distinguished:

- Ensure that the tabletop does not move beyond its vertical and horizontal end positions.
- Prevent collisions of the tabletop with the magnet.
- Define the conditions for manual and automatic movements of the tabletop.
- Enable the operator to control the system by means of the manual button and the tumble switch.

The components and requirements described above are modeled by automata. The plant model consists of 672 states and 14.384 transitions. The total requirement model consists of 4.128 states and 34.884 transitions. Based on these models, the centralized supervisor was synthesized using the TCT tool of [36]. The automaton representing the supervisor consists of 2.976 states and 22.560 transitions. The plant under control of the supervisor was validated using simulation and the supervisor was tested on the real system. Subsequently, the experiment was extended to investigate system evolvability in case of changing functionality. In the original setup, if the table is not maximally up and not maximally out it does not move when the user switches the tumble switch up. As this behavior appeared counter-intuitive for the user, a new specification was introduced stating that when the tumble switch is up, the

table should first go maximally out and then move up if the tumble switch is still up. This functional change introduced only a few small changes to formal requirements and a new supervisor was successfully synthesized. Approximately, four-hour work was needed to produce this new result. Implementing the same functional change in the existing design process was estimated for a week, approximately.

In a related project, supervisor synthesis has also successfully been applied to the patient communication system of the MRI scanner.

2.5.2 Océ printer

In the Océ printer case study, reported in [18], supervisory control synthesis is applied to the scheduling of maintenance operations in a high-end printer of Océ-Technologies (http://www.oce.com) schematically shown in Fig. 2.13. Such a printer consists of three modules: module 1 takes care of paper input, module 2 performs the printing, and module 3 provides stacking and stapling capabilities. The printing process consists of several components as depicted in Fig. 2.13, part *2b*. In this process, the toner image is applied onto the toner transfuse belt and it is fused onto a paper sheet. To maintain high printing quality, several maintenance operations need to be carried out, such as

1. Toner transfuse belt jittering, which displaces the transfuse belt to prolong its lifespan.
2. Black image operation, which prints completely black pages to remove paper dust from the toner transfuse belt.
3. Coarse toner particles removal operation, which removes leftover particles from the toner.

In general, a maintenance operation is scheduled after a given number of prints, unless there is an active print job. There is also a hard threshold of the number of prints after which the maintenance operation must be carried out even if an active print job needs to be interrupted. To perform a maintenance operation, the printing process has to change the power mode: from Run mode used for printing to Standby mode required for maintenance. The change of the power mode can trigger pending

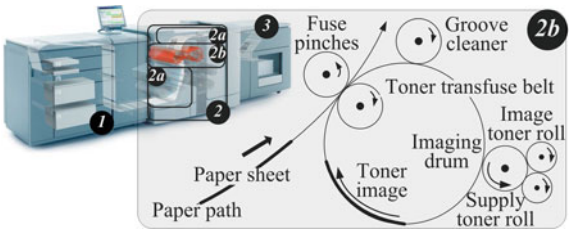

Fig. 2.13 High-end printer

maintenance operations, which might prolong the waiting time of the interrupted print job.

The purpose of control requirements is to coordinate the maintenance procedures with the rest of the printing process. Specifically,

- Maintenance operations may only be performed if the power mode of the printing process is Standby.
- Maintenance operations should be scheduled if their soft deadline is reached and no print jobs are in progress or if their hard deadline is reached.
- Only scheduled maintenance operations can be started.
- The power mode of the printing process should conform to the mode determined by the print job managers unless it is overridden by a pending maintenance operation.

The system components are modeled by automata placed in a state tree structure as defined in [16]. To simplify the specification process, the requirements related to coordination and scheduling of maintenance operations are modeled by generalized state-based expressions. The plant model consists of 25 automata with 2–24 states. The requirements are represented by 23 generalized state-based expressions, which translate to more than 500 standard state-based expressions. Using the synthesis tool of [16], the centralized supervisor is derived. To analyze the controlled behavior, both the plant model and the supervisor are converted to an executable simulation model. After simulation-based validation, the supervisor is converted to C++ for execution on the existing control platform.

In another project, described in [5], supervisor synthesis has also successfully been applied to exception handling in printers.

2.5.3 Theme Park Vehicle

In the theme park vehicle case study, reported in [12], supervisory control synthesis is applied to the multimovers of ETF (http://www.etf.nl), as shown in Fig. 2.14a. Multimovers are automated guided vehicles that can follow an electrical wire integrated in the floor. This track wire produces a magnetic field that can be measured by track sensors. Next to the track wire, floor codes are positioned that can be read by a metal detector. These floor codes give additional information about the track, such as the start of a certain scene program, a switch, a junction, or a dead end. The scene program, which is read by the scene program handler, defines when the vehicle should ride at what speed, when it should stop, rotate, play music and in which direction the vehicle should move if it has arrived at a junction.

An operator is responsible for powering up the vehicle and deploying it into the ride manually. The operator also controls the dispatching of the vehicles in the passenger boarding and outboarding area. The vehicle can receive messages from Ride Control. Ride Control coordinates all vehicles and sends start/stop commands to these vehicles. This communication is either wireless or by means of the track wire.

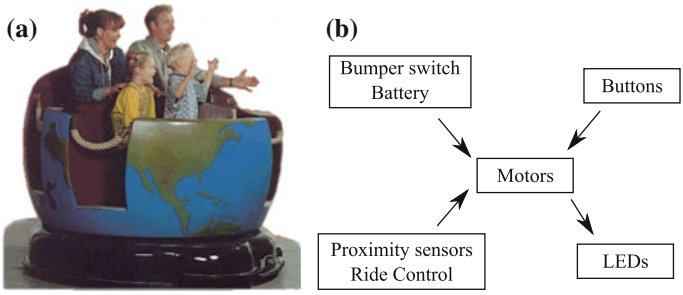

Fig. 2.14 Theme park vehicle. **a** The multimover. **b** Interaction between components

Multimovers are not able to communicate with other vehicles. As safety is an important aspect of this vehicle, several sensors are integrated in it to avoid collisions. First, proximity sensors are integrated in the vehicle to avoid physical contact with other objects. Two types of proximity sensors are used: long-range proximity sensors to detect obstacles in the vicinity of six meters and short-range proximity sensors to detect obstacles in the vicinity of one meter. Second, a bumper switch is mounted on the vehicle that can detect physical contact with other objects. The interactions between vehicle components are shown schematically in Fig. 2.14b.

The main requirement for supervisory control synthesis is safety. Three safety-related aspects can be distinguished:

- **Proximity handling** The supervisor has to assure that the multimover does not collide with other vehicles or obstacles. To this end, proximity sensors are integrated at the front and back which can detect an obstacle in the vicinity of the multimover. To avoid collisions, the multimover should drive at a safe speed and stop if the obstacle is too close to it.
- **Emergency handling** The system should stop immediately and should be powered off when a collision occurs. To detect collisions, a bumper switch is mounted on the multimover. The same applies when the battery level is too low. The Light Emitting Diode (LED) interface should give a signal when an emergency stop has been performed. The multimover should be deployed back into the ride by an operator manually.
- **Error handling** When a system failure occurs (e.g., a malfunction of a motor), the system should stop immediately and should be powered off to prevent any further problems. The LED interface should give a signal that an emergency stop has been performed. The multimover should be deployed back into the ride by an operator manually.

The multimover control problem cannot be solved in the centralized event-based setting because of the large state-space size. To enable the application of distributed synthesis methods, it is divided into five smaller subproblems that are solved separately.

- **LED actuation** An operator must be able to check in which state the multimover is by looking at the interface LEDs. This means that the states of the LEDs represent the current state of the multimover. It is a task of the supervisor to actuate the LEDs according to the state of the multimover.
- **Motor actuation** The drive motor, steer motor, and scene program handler have to be switched on and off according to the state of the multimover. If the multimover is in the state **Active**, all motors can be switched on. If the multimover is in the state **Reset** or **Emergency**, all motors have to be switched off.
- **Button handling** The user interface of the multimover contains three buttons. The reset button is used to reset the vehicle if the multimover is active and deployed into the ride or it is in the state **Emergency**. The forward and the backward buttons are used to deploy the vehicle into the corresponding direction. The supervisor has to assure that the corresponding state is reached after a button is pushed.
- **Proximity and Ride Control handling** On each side of the multimover, two proximity sensors are mounted: one long range and one short range. If a long-range proximity sensor detects an object in the traveling direction, the multimover should react by slowing down to a safe driving speed. If an obstacle is detected by a short-range proximity sensor, the multimover should stop in order to prevent a collision. When the short-range proximity sensor does not detect an object any more, the vehicle should start moving automatically. If the multimover receives a stop command from Ride Control, it should stop as in the case of short-range proximity handling. If Ride Control sends a start command, the multimover should automatically start with the speed depending on the state of the proximity sensors related to the current driving direction.
- **Emergency and error handling** In order to guarantee the safety of the passengers, the multimover should be deactivated immediately when an emergency situation occurs. It should not be possible to reset the multimover if the bumper switch is still activated or the battery power is still too low. A control task of the supervisor is to enter the **Emergency** state of the multimover when an emergency situation occurs.

The plant model is defined by automata representing an event-based abstraction of the actual behavior of the physical components and their resource control. Requirements described above are also defined by automata. Based on the plant and requirement models, an optimal supervisor is synthesized, validated, and implemented. The plant model consists of 17 automata with 2–4 states. The requirements are represented by 30 automata with 2–7 states.

Both a centralized supervisor and a distributed supervisor are synthesized for the supervisory control problem of the multimover. A centralized supervisor has been synthesized with the state-based framework based on state tree structures of [16]. The state-based synthesis produces binary decision diagrams (BDD) for each controllable event. The maximum BDD size is 15 and the minimum BDD size is 1. Furthermore, a distributed supervisor has been synthesized with an aggregated approach of [30]. The results of the aggregated synthesis are shown in Table 2.1.

Table 2.1 Distributed supervisors

Module	Order	# States	# Trans.	Order	# States	# Trans.
LED actuation	1	25	77	5	41	125
Motor actuation	2	41	222	2	257	1428
Button handling	3	465	3477	4	177	765
Emergency handling	4	89	626	3	118	609
Proximity handling	5	225	1953	1	481	4513

Synthesized supervisors were evaluated to check whether the models of the controlled system are consistent with the intended behavior. For this purpose, discrete-event simulation was used persistently. Specifically, the state-space stepper was used to check whether the supervisor disables the right transitions in the right states when evaluating the closed-loop system behavior. The toolset described in [1] was used for discrete-event simulation. Currently, a new version of the toolset supporting MBSE and synthesis is being developed based on CIF [21, 33].

To deliver a proof of concept for synthesis-based engineering, a prototype of a supervisory controller with the synthesized supervisors is implemented in the existing control software of the multimover. This implementation is first tested by means of simulation and then on the existing implementation platform. Subsequently, the number of proximity sensors is extended. The engineering process used presently requires approximately 2 days for making necessary changes to the control system. The synthesis-based engineering process described in this paper requires approximately four hours to cope with the same change.

2.6 Conclusions

Formal models and associated model transformations provide consistent support for synthesis-based engineering. If consequently used from the beginning of the product development process, they constitute a vehicle that:

- Helps in specifying component and system behavior in a structured, systematic, and unambiguous way.
- Forces to clarify all relevant aspects of the system being developed.
- Shortens the time needed to deliver a good design and correctly functioning control software.

The synthesis phase requires modeling of the uncontrolled system by finite-state machines and modeling of the requirements by finite-state machines and logical expressions. This kind of modeling is intuitive and usually results in (relatively) simple and easy to maintain models as input to the synthesis procedure. Because the synthesis is automatic, the focus in the control software development process changes

from designing and debugging controller code to designing and analyzing requirements, if the input models are correct. The downside is, however, that there is not sufficient tool support for validation of these models. This is especially problematic for larger applications. In the industrial case studies described in this paper, simulation was used extensively for this purpose, which consumes too much development time. In a subsequent project, another analysis techniques, like model checking, are investigated in this context.

The results of the case studies show another advantage of synthesis-based engineering that manifests itself when the structure or the requirements of the system under development change due to customer demands. In traditional engineering, all changes have to be made in the software design manually, which is difficult to do without introducing errors or inconsistencies. In the synthesis-based approach, only component models or requirement models have to be adapted and a new supervisor can again automatically be synthesized. Since the desired behavior is specified in models instead of in the control software, it is easier to validate it with respect to the original informal specifications. This also contributes to system evolvability.

References

1. Baeten, J., van Beek, D., Cuijpers, P., Reniers, M., Rooda, J., Schiffelers, R., Theunissen, R.: Model-based engineering of embedded systems using the hybrid process algebra Chi. ENTCS **209**, 21–53 (2008)
2. Baeten, J., van Beek, D., Luttik, B., Markovski, J., Rooda, J.: A process-theoretic approach to supervisory control theory. In: Proceedings of ACC, pp. 4496–4501. IEEE (2011)
3. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
4. van Beek, D., Man, K., Reniers, M., Rooda, J., Schiffelers, R.: Syntax and consistent equation semantics of hybrid Chi. JLAP **68**(1–2), 129–210 (2006)
5. Bertens, E., Fabel, R., Petreczky, M., van Beek, D., Rooda, J.: Supervisory control synthesis for exception handling in printers. In: Proceedings of the Philips Conference on Applications of Control Technology (2009)
6. Braspenning, N., Boumen, R., van de Mortel-Fronczak, J., Rooda, J.: Estimating and quantifying the impact of using models for integration and testing. Comput. Ind. **62**(1), 65–77 (2011)
7. Braspenning, N., van de Mortel-Fronczak, J., Rooda, J.: A model-based integration and testing method to reduce system development effort. ENTCS **164**(4), 13–28 (2006)
8. Brinksma, E., Tretmans, J.: Testing transition systems: an annotated bibliography. In: Modeling and Verification of Parallel Processes, Lecture Notes in Computer Science, vol. 2067, pp. 187–195. Springer (2001)
9. Cai, K., Wonham, W.: Supervisor localization: a top-down approach to distributed control of discrete-event systems. IEEE Trans. Autom. Control **55**(3), 605–618 (2010)
10. Estefan, J.: Survey of Model-Based Systems Engineering (MBSE) methodologies. Technical report, INCOSE (2008). http://www.incose.org
11. Flordal, H., Malik, R., Fabian, M., Akesson, K.: Compositional synthesis of maximally permissive supervisors using supervisor equivalence. Discrete Event Dyn. Syst. **17**(4), 475–504 (2007)
12. Forschelen, S., van de Mortel-Fronczak, J., Su, R., Rooda, J.: Application of supervisory control theory to theme park vehicles. Discrete Event Dyn. Syst. **22**(4), 511–540 (2012)
13. Hill, R., Tilbury, D., Lafortune, S.: Modular supervisory control with equivalence-based conflict resolution. In: Proceedings of ACC, pp. 491–498 (2008)

14. Leduc, R., Lawford, M., Wonham, W.: Hierarchical interface-based supervisory control-part II: parallel case. IEEE Trans. Autom. Control **50**(9), 1336–1348 (2005)
15. Liu, J., Darabi, H.: Ramadge-Wonham supervisory control of mobile robots: lessons from practice. Proc. ICRA **1**, 670–675 (2002)
16. Ma, C., Wonham, W.: Nonblocking supervisory control of state tree structures. IEEE Trans. Autom. Control **51**(5), 782–793 (2006)
17. Malik, R., Flordal, H.: Yet another approach to compositional synthesis of discrete event systems. In: Proceedings of WODES, pp. 16–21 (2008)
18. Markovski, J., Jacobs, K., van Beek, D., Somers, L., Rooda, J.: Coordination of resources using generalized state-based requirements. In: Proceedings of WODES, pp. 297–302 (2010)
19. Martin, J.: Systems Engineering Guidebook. CRC Press (1996)
20. Muller, G.: Coupling enterprise and technology by a compact and specific architecture overview. In: Proceedings of INCOSE (2007)
21. Nadales Agut, D.E., van Beek, D.A., Rooda, J.E.: Syntax and semantics of the compositional interchange format for hybrid systems. JLAP **82**(1), 1–52 (2012)
22. Noorbakhsh, M., Afzalian, A.: Design and PLC based implementation of supervisory control for under-load tap-changing transformers. In: Proceedings of ICCAS, pp. 901–906 (2007)
23. Nourelfath, M., Niel, E.: Modular supervisory control of an experimental automated manufacturing system. Control Eng. Pract. **12**(2), 205–216 (2004)
24. van Osch, M.: Automated model-based testing of hybrid systems. Ph.D. thesis, Eindhoven University of Technology, The Netherlands (2009)
25. de Queiroz, M., Cury, J.: Modular supervisory control of composed systems. In: Proceedings of ACC, pp. 4051–4055 (2000)
26. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987)
27. Rechtin, E., Maier, M.: The Art of Systems Architecting. CRC Press (1997)
28. Schiffelers, R., Theunissen, R., van Beek, D., Rooda, J.: Model-based engineering of supervisory controller using CIF. In: Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling. Electronic Communication of the EASST, vol. 21, pp. 1–10 (2009)
29. Su, R., van Schuppen, J., Rooda, J.: Synthesize nonblocking distributed supervisors with coordinators. In: Proceedings of the 17th Mediterranean Conference on Control and Automation, pp. 1108–1113 (2009)
30. Su, R., van Schuppen, J., Rooda, J.: Aggregative synthesis of distributed supervisors based on automaton abstraction. IEEE Trans. Autom. Control **55**(7), 1627–1640 (2010)
31. Su, R., van Schuppen, J., Rooda, J.: The synthesis of time optimal supervisors by using heaps-of-pieces. IEEE Trans. Autom. Control **57**(1), 105–118 (2012)
32. Su, R., Thistle, J.: A distributed supervisor synthesis approach based on weak bisimulation. In: Proceedings of WODES, pp. 64–69 (2006)
33. Systems Engineering Group TU/e: CIF toolset. http://cif.se.wtb.tue.nl (2013)
34. Theunissen, R., Schiffelers, R., van Beek, D., Rooda, J.: Supervisory control synthesis for a patient support system. In: Proceedings of ECC, pp. 4647–4652 (2009)
35. Wong, K., Wonham, W.: Hierarchical control of discrete-event systems. Discrete Event Dyn. Syst.: Theory Appl. **6**(3), 241–273 (1996)
36. Wonham, W.: Supervisory control of discrete-event systems. Technical report, University of Toronto (2010)