# Chapter 49
# Coordination Control of Complex Machines

J.C.M. Baeten, D.A. van Beek, J. Markovski, L.J.A.M. Somers

**Abstract** Control and coordination are important aspects of the development of complex machines due to an ever increasing demand for better functionality, quality, and performance. In WP6 of the C4C project we developed a synthesis-centric systems engineering framework suitable for supervisory coordination of complex systems. The framework was employed to synthesize and validate a supervisory coordinator for maintenance procedures for a prototype of a high-tech Océ printer, showing proof of concept and viability of the proposed framework. The supervisor eliminates undesired behavior that could occur as a result of undesired interaction of the distributed printer components. In this chapter, we discuss the model-based systems engineering framework that was employed for synthesis of the supervisor and we illustrate the modeling process.

## 49.1 Background and Motivation

In the last few decades, control software development has taken a more central role due to the ever increasing complexity of the machines, demands for higher quality and performance, and improved safety and ease of use [5]. Traditionally, the control software requirements are formulated informally in some sort of specification documents by domain engineers, to be translated into control software by software engineers. This is a time-consuming and an error prone process, since control requirements are often ambiguous and change frequently, so the produced software needs to be validated against the machine. If validation fails, the code must be rewritten leading to a *define-validate-redefine loop*.

Jos C.M. Baeten · Bert (D.A.) van Beek · Jasen Markovski · Lou J.A.M. Somers
Department of Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513, 5600MB, Eindhoven, The Netherlands,
e-mail: j.c.m.baeten@tue.nl,d.a.v.beek@tue.nl,j.markovski@tue.nl,l.j.a.m.somers@tue.nl

This issue in control software design gave rise to supervisory control theory [11, 2, 3, 6], where models of high-level supervisory controllers, referred to as *supervisors*, are synthesized automatically based upon a formal model of the uncontrolled system, known as *plant*, and a model of *the control requirements*. Supervisory controllers observe the discrete-event behavior of the system and make control decisions based on the observed information. The supervisory controller synthesis problem is to achieve the greatest possible allowed behavior, which is specified by the control requirements, based on a given observable behavior.

By applying automated control software generation based upon formal behavioral models, we aim to:

1. Shift the focus of software engineers from writing and debugging code to modeling the plant and the control requirements.
2. Increase the use of models. Software engineers need not think in terms of code, since it is generated automatically. Instead, they make the formal models in terms of behavior, thus improving their communication with the domain engineers and other involved parties.
3. Shorten the design loop. Adjustments in the control specifications result in model changes, followed by automated synthesis of control software, which is likely to be less time consuming than directly adapting control code.
4. Use the formal behavioral models for simulation. We can validate the supervisor before expensive prototypes are built increasing confidence in the design. It also provides opportunity for verification, performance analysis, and reliability analysis.
5. Reuse of models and improved evolvability of the process. When developing a new product, models can be reused more easily than specific code since small adjustments are incorporated more easily. Furthermore, the changes in the control requirements are more easily managed as we directly synthesize provably correct models.

Reflecting on the points above we aim to: increase product quality by dealing with increased machine complexity more easily; reduce costs by validating controllers before expensive prototypes are in place; reduce time-to-market by improving communication between the engineers and shortening the design loop.

## 49.2 Model-Based Systems Engineering Framework

To structure the process of supervisory control synthesis we employ the framework depicted in Fig. 49.1 [12, 9], which is a refinement and extension of the framework defined in [1]. Domain engineers define the desired overall system specification (Specification *Controlled System* in Fig. 49.1), that is later elaborated into a design document (Design *Controlled System*) by domain and software engineers together. The design defines the architecture of the system, and its composition into subsys-

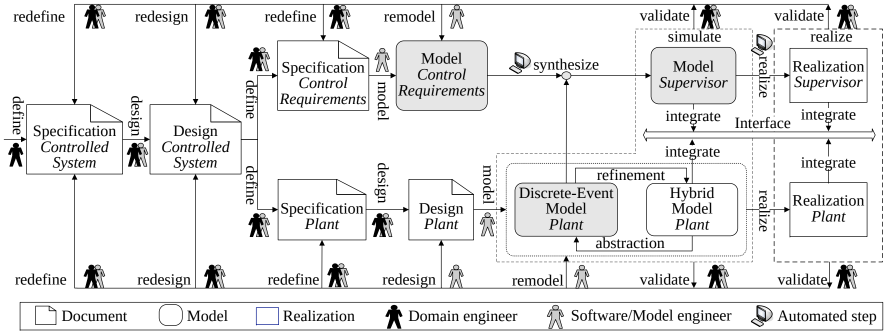

**Fig. 49.1** Model-based systems engineering framework for supervisory control synthesis.

tems. In this case, we consider as subsystems the controller and uncontrolled system, the latter also referred to as ‘plant’.

The subsystems are likewise (informally) specified, resulting in the documents Specification *Control Requirements* and Specification *Plant*, in Fig. 49.1. The supervisory control synthesis framework then facilitates formal specification of control requirements (Model *Control Requirements*), instead of specification of a controller. Supervisory control synthesis also requires a discrete-event model of the uncontrolled system (plant).

Plants typically contain hybrid (continuous and discrete-event) behavior suitable for simulation. For synthesis purposes [11, 2], continuous behavior is abstracted from, i.e., a pure high-level discrete-event model is derived. Alternatively, a discrete-event model can be made and subsequently refined to a hybrid model.

The synthesis algorithm generates a minimally-restrictive supervisor, based on the models of the control requirements and plant. Such a supervisor, by construction satisfies the control requirements and is nonblocking, which means that from any reachable state, always a marked state can be reached. Many different kinds of supervisory control synthesis algorithms exist, see for example [11, 2, 7, 13].

The model of the generated supervisor can then be combined with the (hybrid) model of the plant, synchronizing on events and location names, for early, model-based validation of the controlled system. Such a coupling is possible as the supervisor only considers the discrete-event behavior of the system, which is orthogonal to the continuous dynamics. This validation should ensure that the observed controlled system behavior satisfies the system behavior and control requirements as specified in the informal documents of Fig. 49.1. If validation fails, remodeling the control requirements or even complete revision proves necessary.

As a last step, the control software is generated automatically from the validated models. Our framework also supports other options for early integration, such as hardware-in-the-loop-simulation, by coupling of the realization of the supervisor with the hybrid real-time simulation model of the plant.

The tooling used for supervisory control synthesis was based on the first version of the Compositional Interchange Format, CIF 1 [15]. For simulation-based valida-

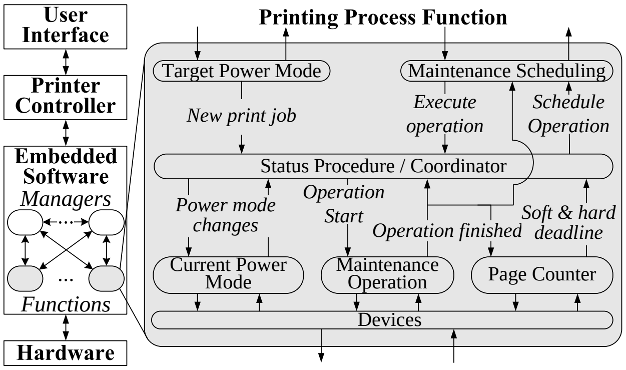

**Fig. 49.2** Printing process function and the corresponding plant

tion, CIF 2 [10] was used. CIF is an expressive formalism, with a formal semantics, based on hybrid automata. CIF interfaces with many different tools by means of model transformations. Recently, the formal semantics and modeling concepts have been considerably simplified, resulting in CIF 3 [14]. The CIF toolset is now implemented in Eclipse [4], using Java as implementation language, and SVG (Scalable Vector Graphics) [16] for interactive, simulation based visualization. Recent improvements to the CIF 3 development environment are an Eclipse editor with integrated real-time background parsing and type checking. Furthermore, the CIF 3 simulator has been redesigned for fast simulation by employing run-time code generation.

## 49.3 Case Study: High-Tech Printers

We illustrate the modeling process on a case study involving coordination of maintenance procedures of a printing process of a high-end Océ printer of [8]. Due to confidentiality concerns, we can only present an obfuscated part of the case study.

We abstractly depict a printing process function in Fig. 49.2, where the control architecture of the printer is given to the left. We coordinate the function responsible for the maintenance of the printing process. The printer executes print jobs in run mode of operation, whereas several maintenance operations to preserve print quality, have to be carried out in standby mode. Maintenance operations are scheduled based on the amount of pages printed since the last maintenance. Soft deadlines denote that a maintenance *can* be scheduled and hard deadlines denote that the maintenance *must* be scheduled. Maintenance procedures with expired soft deadlines can be postponed if there is an ongoing print job, but hard deadlines must be respected.

A printing process function comprising one maintenance operation is depicted in Fig. 49.2. The supervisory control problem is to synthesize a model of the Status Procedure, which is responsible for coordinating the other procedures given input

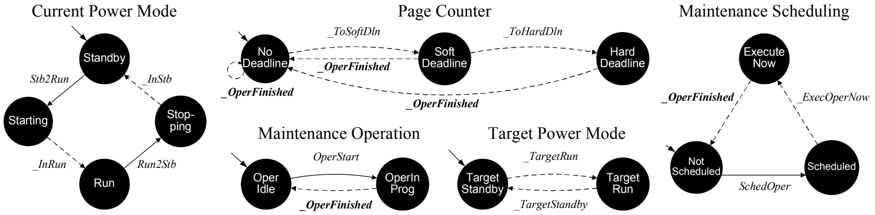

**Fig. 49.3** Printing process function and the corresponding plant

from the controllers. The plant that models the printing process function is given in Fig. 49.3. For modeling, we employ state-labeled finite automata. The state labels are employed to keep track of the state of the plant, and can be referenced in the control requirements. We employ two types of state-based control requirements [9]: $\phi$ and $\xrightarrow{e} \implies \phi$, for some some event $e$, and some Boolean formula $\phi$ over the state labels, using the logical operators $\wedge$, $\vee$, $\neg$, and $\implies$. The first requirement type specifies an invariant $\phi$ that must hold for every state of the supervised system, whereas the second type specifies necessary conditions, given by $\phi$, for enabling event $e$.

Uncontrollable events are underscored. Initial states have incoming arrows, whereas marked states, which specify states in which the systems is considered to have successfully completed some task [2], coincide with the initial states, as we are dealing with a reactive system. The plant is formed by the synchronization of the automata in Fig. 49.3. Current Power Mode sets the power mode to run or standby, using *Stb2Run* or *Run2Stb*, respectively, and sends back feedback by employing *_InRun* and *_InStb*, respectively. Maintenance Operation either carries out a maintenance operation, started by *OperStart* or it is idle. The confirmation is sent back by the event *_OperFinished*, which synchronizes with Maintenance Scheduling and Page Counter. Page Counter announces when soft or hard deadlines are reached using *_ToSoftDln* and *_ToHardDln*, respectively. The page counter is reset, triggered by the synchronization on *_OperFinished*, each time maintenance is finished. The controller Target Power Mode sends signals regarding incoming print jobs to Status Procedure. The event *_TargetRun* should set the printing process to run mode for printing. When the print job is finished, the event *_TargetStandby* is activated. Maintenance Scheduling receives a request for maintenance with respect to expiration of Page Counter from Status Procedure, by the event *SchedOper*, and forwards it to the manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure, using *_ExecOperNow*. It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling, again triggered by *_OperFinished*.

The coordination is performed according to the following requirements: (1) Maintenance operations can be performed only when Printing Process Function is in standby; (2) Maintenance operations can be scheduled only if a soft deadline has been reached and there are no print jobs in progress, or a hard deadline has passed;

(3) Only scheduled maintenance operations can be started; (4) The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation. For a detailed account of the model-based systems engineering process and specification and formalization of the control requirements, we refer to [8].

(1) To model this requirement, we consider the states from Current Power Mode and Maintenance Operation and we require that it must always hold (R1) $\mathit{OperInProg} \Rightarrow \mathit{Standby}$.

(2) The states labeled by *SoftDeadline* and *HardDeadline* indicate when soft and hard deadline is reached, respectively. State *TargetRun* of Target Power Mode states that there is a print job in progress. The event *SchedOper* is responsible for scheduling maintenance procedures. We specify the requirement as (R2) $\xrightarrow{\mathit{SchedOper}} \Rightarrow (\mathit{SoftDeadline} \wedge \neg \mathit{TargetRun}) \vee \mathit{HardDeadline}$.

(3) The maintenance operation can be started when the maintenance scheduling is completed, which is modeled as (R3) $\xrightarrow{\mathit{OperStart}} \Rightarrow \mathit{ExecuteNow}$.

(4) The last condition is modeled by two separate requirements for switching from Run to Standby mode, and vice versa. We can change from run to standby mode if this is required by the manager, i.e., identified by state *TargetRun*, and there is no need to start a maintenance operation, identified by $\neg \mathit{ExecuteNow}$. The transitions labeled by *Stb2Run* are enabled by (R4) $\xrightarrow{\mathit{Stb2Run}} \Rightarrow \mathit{TargetRun} \wedge \neg \mathit{ExecuteNow}$. In the other direction, we have (R5) $\xrightarrow{\mathit{Run2Stb}} \Rightarrow \mathit{TargetStandby} \vee \mathit{ExecuteNow}$.

Finally, we synthesize a supervisor by employing the plant of Fig. 49.2 and the control requirements (R1) – (R5).

## 49.4 Research Contributions

The study of the informal specification documents revealed that engineers use a state-based approach, i.e., they give relations when a certain activity may be performed with respect to the state of the machine. Unfortunately, the corresponding synthesis tool [6] requires as input the exact opposite, i.e., the modeler has to specify which behavior is undesired, in the form of a negation of a conjunction of automaton locations. The latter leads to less intuitive specifications and results in a large number of control requirements.

To improve the modeling process and support greater modeling convenience, we generalize the control requirements to enable unrestricted use of propositional logic that we found in the specification documents. Thereafter, we automatically translate the generalized control requirements to an input suitable for the synthesis tool, which is a structurally restricted conjunctive normal form of the control requirements. This enabled us to specify the complete set of coordination rules for the printing process. For a detailed discussion of this transformation, including a formal definition, proof of correctness, and details on the implementation, we refer the interested reader to [9]. As an illustration of the effectiveness of our method, a part of the case study modeled by 23 generalized control requirements, resulted in 500+ requirements in the original form. Admittedly, if one were to model the system in the original setting, a more optimal approach might have been possible, but then the modeling process would be to cleverly specify coordination rules, in an unintuitive form, which is both a time-consuming and an error-prone modeling process.

The issues of Fig. 49.2 were resolved by the supervisory controller, validated by means of simulation, and tested via a prototype implementation of the control software. Additionally, we could specify forms of state-based expressions which do not fit the supervisory control format: instead of state-event *exclusion* expressions of the form $\xrightarrow{e} \implies \phi$, we also used state-event *inclusion* expressions of the form $\phi \implies \xrightarrow{e}$. Since $\xrightarrow{e} \implies \phi$ is equivalent to $\neg\phi \implies \not\xrightarrow{e}$, the state-event exclusion predicate defines a state $\neg\phi$ where the event $e$ is disabled, whereas the state-event inclusion defines a state where the event is enabled. The state-event inclusion predicates can be interpreted as verification properties of plant functionalities, which enables us to combine synthesis and verification. Ongoing research investigates the use of so-called reactive supervisor synthesis, that aims to verify that desired behavior will be present in the supervised system during the synthesis procedure. Namely, supervisor synthesis caters only for safety properties, whereas we aim to guarantee progress properties of the supervised system, which ensure that the desired functionality is preserved.

We find that employing formal models is a key element for successful application of a synthesis-centric systems engineering process. Model-based specifications are consistent and less ambiguous than informal specification documents, forcing the engineers to clarify all aspects of the system. The proposed framework most importantly affects the control software development process, switching the focus from interpreting requirements, coding, and testing, to analyzing requirements, modeling, and validating the behavior of the system.

## 49.5 Further Research

Despite being able to show a proof of concept and to generate control software for a prototype of a future high-tech printer, the proposed approach needs further improvement, and we foresee the need for advancement in several important aspects. Techniques are needed that can directly synthesize supervisors for plants incorporating data, somewhat mitigating the state explosion problem. The control requirements should be reinforced with specific efficiently-computable liveness or progress properties, related to the aforementioned reactive supervisor synthesis. Finally, an investigation is needed into suitable software and hardware architectures for automatic control software synthesis (as existing implementations vary per case), fitting the synthesized software in existing environments tailored for manual control software development. By working on and answering the above challenges, we hope to provide valuable model-based development techniques and tools for the software engineers of the future.

## References

1. N. C. W. M. Braspenning, J. M. van de Mortel-Fronczak, and J. E. Rooda. A model-based integration and testing method to reduce system development effort. *Electronic Notes in Theoretical Computer Science*, 164:13–28, 2006.
2. C. Cassandras and S. Lafortune. *Introduction to discrete event systems, 2nd edition*. Springer, 2008.
3. W. M. Wonham. Supervisory control of discrete-event systems. Dept. Elect. Comput. Eng. Univ. Toronto, http://www.control.toronto.edu/DES/, 2012.
4. The Eclipse Foundation. Eclipse. http://www.eclipse.org/, 2013.
5. N. G. Leveson. The challenge of building process-control software. *IEEE Software*, 7(6):55–62, 1990.
6. C. Ma and W. M. Wonham. *Nonblocking Supervisory Control of State Tree Structures*, volume 317 of *Lecture Notes in Control and Information Sciences*. Springer, 2005.
7. Chuan Ma and W. M. Wonham. Nonblocking supervisory control of state tree structures. *IEEE Transactions on Automatic Control*, 51(5):782–793, 2006.
8. J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. A. M. Somers, and J. E. Rooda. Coordination of resources using generalized state-based requirements. In *Proceedings of WODES 2010*, pages 300–305. IFAC, 2010.
9. J. Markovski, D. A. van Beek, R. J. M. Theunissen, K. G. M. Jacobs, and J. E. Rooda. A state-based framework for supervisory control synthesis and verification. In *Proceedings of CDC 2010*, pages 3481–3486. IEEE, 2010.
10. D. E. Nadales Agut, D. A. van Beek, and J. E. Rooda. Syntax and semantics of the compositional interchange format for hybrid systems. *Journal of Logic and Algebraic Programming*, 82(1):1–52, 2013.
11. P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.
12. R. R. H. Schiffelers, R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda. Model-based engineering of supervisory controllers using CIF. In *Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling*, volume 21 of *Electronic Communications of the EASST*, pages 1–10, Denver, 2009.
13. Rong Su, J. H. van Schuppen, and J. E. Rooda. Aggregative synthesis of distributed supervisors based on automaton abstraction. *IEEE Transactions on Automatic Control*, 55(7):1627–1640, 2010.
14. Systems Engineering Group TU/e. CIF toolset. http://cif.se.wtb.tue.nl, 2013.
15. D. A. van Beek, M. A. Reniers, R. R. H. Schiffelers, and J. E. Rooda. Foundations of an interchange format for hybrid systems. In Alberto Bemporad, Antonio Bicchi, and Giorgio Butazzo, editors, *Hybrid Systems: Computation and Control, 10th International Workshop*, volume 4416 of *Lecture Notes in Computer Science*, pages 587–600, Pisa, 2007. Springer-Verlag.
16. W3C. W3C SVG working group. http://www.w3.org/Graphics/SVG/, 2013.