Science of Computer Programming 97 (2015) 17–22

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

The role of supervisory controller synthesis in automatic control software development

Jos Baeten [a,b,1], Jasen Markovski [b,*,1]

[a] Centrum Wiskunde & Informatica, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
[b] Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

HIGHLIGHTS

- We discuss the role of supervisor synthesis in automated software code generation.
- The proposed approach is systematic and based on process theory.
- We implemented a model-based systems and software engineering framework.
- The framework has been applied to multiple industrial studies.

ARTICLE INFO

Article history:
Received 26 September 2013
Accepted 5 November 2013
Available online 21 November 2013

This article is dedicated to Paul Klint, an exemplary computer scientist, who always manages to combine sound theoretical foundations and practical implementations in software

Keywords:
Supervisory control theory
Model-driven development
Software synthesis

ABSTRACT

We give an overview of a model-driven systems engineering approach for high-tech systems that relies on supervisory controller synthesis. The proposed framework has a process-theoretic foundation and supports extensions with quantitative features. We briefly discuss several industrial case studies that highlight the advantages of the proposed approach.

1. Model-driven control software development

Embedding information and communication technology in consumer and industrial products was enabled by advances in software that carries most of these products' functionalities. The need for development techniques that can guarantee software quality is more than apparent. Conferences like Software Development Automation, Model-Driven Engineering Languages and Systems, or Applications of Concurrency in System Design, are some of the venues where this need has been recognized and studied by means of automated model-driven software development techniques. In this paper, we would like to point to one model-driven systems engineering approach, referred to as supervisory controller synthesis, which targets discrete-event control software for high-tech and complex systems. We find this approach to be relevant to the software development community and we hope that it might offer novel insights in development of quality control software, and bring the communities of software development, systems engineering, and formal methods closer together.

* Corresponding author.
E-mail addresses: jos.baeten@cwi.nl (J.C.M. Baeten), j.markovski@tue.nl (J. Markovski).
[1] Supported by Dutch NWO project: ProThOS, no. 600.065.120.11N124.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.11.016

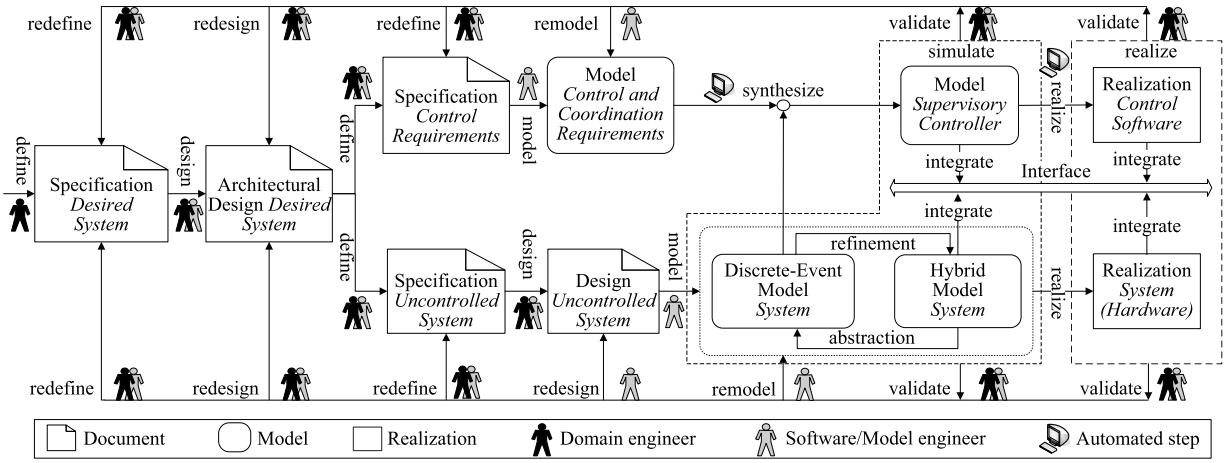

Fig. 1. A synthesis-centric model-driven systems engineering framework.

Development of quality control software is becoming an important challenge in the production process of high-tech complex systems. The ever-increasing complexity, in conjunction with the market pressure for improved quality, safety, ease of use, and performance, promote the former as an important future bottleneck. Many model-driven software development approaches, typically relying on formal methods, are advocated as cost-effective approaches that support rigorous specification, validation, and testing of software [1,2]. Some standards, like [3,4], even make these methodologies compulsory. Admittedly, the use of formal methods requires a substantial effort in the initial development and design phases. However, formal specifications and early model validation greatly decrease the number of errors found during testing and integration phases mitigating overall product-development time and costs, while increasing quality [5].

Unfortunately, there is still insufficient emphasis on the integration of the formal methods in the systems engineering process, e.g., as noted in [6], despite successful industrial cases involving formal verification and validation like [7–9]. Furthermore, it has been noted early on [10] that traditional software development approaches are not entirely adequate to control software development. This is mainly due to the fact that control and coordination requirements, which typically specify safety properties in an informal and often ambiguous manner, frequently change during the design process, inducing a large number of expensive (re)coding–validation–testing iterations. Consequently, partially employing formal techniques for some development phases without supporting the complete design process might not suffice, calling forth a shift from process-based towards model-driven development [11].

2. A synthesis-centric approach

To mitigate some of the above issues in the design and development of supervisory control software for high-tech complex machines, we propose to employ a synthesis-centric model-driven systems engineering framework developed in [12–14], depicted in Fig. 1. Supervisory controllers observe and coordinate the discrete-event behavior of the concurrently running components of the system. Based on the observations made from the uncontrolled system, these controllers make a decision on which activities the system is allowed to perform, and send back control signals that actuate the system. The layer of supervisory control is on a high level of abstraction, residing between the user interaction and the resource (embedded) control of the machine [15]. The framework is synthesis-centric as models of the supervisory controllers are synthesized automatically based on the models of the uncontrolled system and the control requirements.

The modeling process is initiated by domain engineers that propose an informal specification of the desired system. Subsequently, an architectural design of the system is made in cooperation with software engineers, which most importantly defines the control software architecture and modeling level of abstraction. Based on the design, specification and models of the system and the control and coordination requirements are made in parallel. The model of the system comprises the models of its (synchronizing) components, whereas the control requirements specify the allowed (safe) behavior of the system.

Most systems contain mixed continuous and discrete-event or hybrid behavior and hybrid models of the system are made for the purpose of simulation and validation. These models are abstracted to a discrete-event model, required by the synthesis procedure [16,15,17]. The discrete-event model of the uncontrolled system together with the model of the control requirements serve as input for the synthesis of the model of the supervisory controller. The control software is automatically generated based on the synthesized model of the supervisory controller. The framework provides for software and hardware-in-the-loop simulation for validation of the control against the models or the prototypes of the system, respectively.

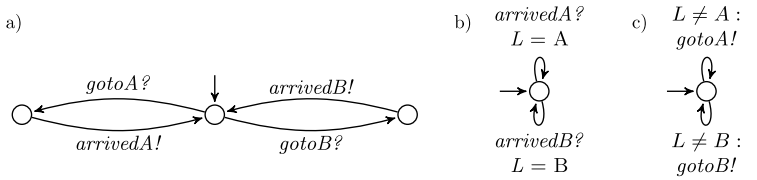

Fig. 2. (a) A plant that models the behavior of an automated guided vehicle; (b) An observer that keeps track of the location of the vehicle of (a); (c) A supervisor that ensures proper coordination of the vehicle of (a).

We note that supervisory controller synthesis is strongly related to the technique of software synthesis [18,19]. However, these two seemingly overlapping techniques differ significantly in their goal and application. Unlike supervisory controller synthesis, that aims to restrict a given finite model of a system such that a set of control requirements is satisfied, software synthesis is generally concerned with synthesis of software or program skeletons and schemes that satisfy a given set of formal specifications. This makes the latter technique more general and the main research seems to focus on decidability results, mainly relying on game-theoretic approaches. Some of the results are applicable in the systems engineering domain, e.g., synthesis of time-optimal schedulers [20]. However, for the most part, the application of these two techniques differs considerably, as the supervisory control community focuses on techniques and approaches viable for solving industrially-relevant problems, often having to compromise between expressivity and efficiency.

3. A process-theoretic foundation

Supervisory control theory [16,15] deals with characterization of the existence and properties of the model of the supervisory controller and its relation with the model of the uncontrolled system and the control requirements. The central notion of controllability is traditionally defined in terms of observed traces and it originally considered only deterministic systems [16,15]. The introduction of internal events, not observable by the supervisory controller led to extensions with nondeterministic behavior.

Several approaches, like [21,22] attempted to employ refinements to capture controllability in a setting with failure traces and trajectories. Subsequent attempts, like [23–25], recognized the importance of (bi)simulation-based approaches. A process-theoretic approach to supervisory control theory was proposed in [26], employing the behavioral preorder partial bisimulation.

This preorder acts as a refinement between the models of the unsupervised and the supervised system. Some activities of the system, like user interaction or sensor observations, are beyond the control of the supervisory controller, whereas the control is exercised by interaction with the actuators of the system. The controllability conditions have to ensure that the supervisory controller does not achieve its objectives by wrongly discarding these uncontrollable events. Roughly speaking, the partial bisimulation preorder states that the model of the uncontrolled system should simulate the model of the controlled system, whereas the uncontrollable events should be bisimulated. This relationship ensures that supervisory controller restricts the original system and it does not ignore any input from the system.

By casting supervisory control theory in a process-theoretic setting, we profit from decades of research in concurrency, being able to extend the theory with states and data [27] or quantitative features, like time and stochastic delays [28], effectively being able to couple supervisory control theory with formal verification and performance analysis.

4. An illustrative example

We give a small example to illustrate the (modeling) process for supervisor synthesis. Let us assume that we have an automated guided vehicle capable of travel to two different locations, say A and B, and initially it can be at some other location, say I. The vehicle can accept two commands to go to the respective location, i.e., *gotoA?* and *gotoB?*. Once the vehicle has reached the corresponding location, it reports back by using the sensor signals *arrivedA!* and *arrivedB!*, respectively. We distinguish between the direction of communicated commands and signals and by *event?* we denote a recipient party of the communication, whereas by *event!* we denote a sender party. We can model the behavior of such a vehicle by the transition system depicted in Fig. 2(a), where the initial state is given by an incoming arrow. We note that the vehicle has no memory regarding its current location, so we cannot demand of it to report back with its location.

Now, let us coordinate the movement of this vehicle by means of a supervisor. The coordination rules require that if the vehicle is at location X, then it should not be given the redundant command that sends the vehicle to the same location X, for $X \in \{\mathrm{A}, \mathrm{B}\}$. By employing only the transition system in Fig. 2(a) such coordination proves to be unexpectedly difficult, more so because the control requirements actually relate to information not directly accessible in the plant. To resolve this issue, it is often required that we augment the model of the system by so-called observer processes, which keep track of some important system behavior. We note that we can also employ a pure discrete-event approach, but then the modeling convenience and the clarity of the specifications will be greatly reduced.

We introduce an observer for the location of the vehicle, as depicted in Fig. 2(b), in order to track the location of the vehicle and employ it for supervision. The observer updates a variable, say L, with respect to the current location of the vehicle, where I is the initial location. The observer waits for the confirmation signal *arrivedX!* for $X \in \{A, B\}$, sent from the vehicles, that it has reached the corresponding location in order to update its status to $L = X$. By employing this location information, the supervisor can make the (correct) decision on which command is allowed to be issued.

An immediate advantage of employing an observer is that we can actually (formally) specify the control requirements directly in terms of the plant and the observed information as: *If $L = X$, then the event* gotoX *should not be enabled*, for $X \in \{A, B\}$. By employing these control requirements, we can synthesize the supervisor depicted in Fig. 2(c). This data observation-based supervisor enables the movement commands, relying on guards $L \neq X$ for $X \in \{A, B\}$, which observe the shared location variable L that is provided by the observer.

5. Industrial application

The model-driven systems engineering framework of Fig. 1 has been successfully applied in several industrial case studies involving movement control of a patient support system for a Philips *MRI scanner* [29], coordination of a printing process of an Océ *High-tech printer* [30], and movement coordination of *Theme park vehicles* [31]. In all cases, the synthesized supervisory controllers were successfully tested using hardware-in-the-loop integration, where the hardware was supervised by employing the model of the supervisory controller for which we used appropriate interfaces with the machines that employ the controller protocols [12].

The goal of the case study *MRI scanner* reported in [29] is safe positioning of patients inside an MRI scanner. To this end, models of the patient support system and the user interface of the machine have been made, leading to $6.3 \cdot 10^9$ states in the unsupervised system. The synthesized controller was correctly generated in the first attempt. Ongoing research focuses on the coordination of the functionality of the whole MRI scanner, employing decentralized supervision that locally coordinates the different components of the machine and synchronizes on the supervision decisions.

The case study *High-tech printer* of [30] deals with coordination of maintenance procedures of the printing process, which applies a toner image onto a paper sheet. To maintain high printing quality, several maintenance operations have to be carried out after a certain number of prints. These maintenance procedures should not interrupt ongoing or pending print jobs whenever possible, but if a given threshold is exceeded, maintenance becomes mandatory to preserve the print quality. The model of the printing process comprises 25 automata with 2 to 24 states. To specify the control requirements succinctly, we employed parameterized state-based control requirements, which induced 500+ logical restrictions as employed by the synthesis tool [32].

In the case study *Theme park vehicles* of [31] we deal with movement coordination of automated guided theme park vehicles such that safety of the passengers is guaranteed. The vehicles can move in multiple directions sensing their position by means of a track wire and they are equipped with short and long range proximity sensors. The control problem is to avoid collision by correct proximity handling. In case of an exception, like depleted battery, the supervisory controller must provide for emergency handling of the other vehicles. The model comprised 17 automata with 2 to 4 states, whereas the control requirements where represented by 30 automata with 2 to 7 states.

6. Concluding remarks

The use of formal models is a key element for successful application of the synthesis-based systems engineering process of Fig. 1. As the software code is generated automatically, software engineers deal with the formal models in terms of behavior, improving their communication with other involved parties. The proposed framework most importantly affects the software development process, switching the focus from interpreting requirements, coding, and testing to analyzing requirements, modeling, and validating the behavior of the system. This increases the quality of the software dramatically, as nearly three quarters of failures found in operational software originate from errors or oversights in the requirements [33]. Admittedly, our approach may prolong the obtention of the initial control design, but by catching design faults early, the testing costs are mitigated and the overall production time is shortened.

Supervisory controller synthesis becomes viable as engineers nowadays are familiar with building models for simulation and validation purposes. The synthesized models provide opportunity for verification, performance, and reliability analysis, increasing the confidence in the control design and validating it before expensive prototypes are built. Finally, the proposed approach greatly increases the evolvability of the system. In the theme park vehicle case study [31], there was a last minute addition of an extra sensor. This mandated the introduction of the model of the new component to the model of the system, adaption of the control requirements, synthesis of a new supervisor, and its validation. The whole procedure was completed within four hours, leading to a viable control design. According to the industrial partner, by employing their software development process, this change in the control software would take about one week.

Additional emerging application of supervisory control is deadlock avoidance in concurrency programming [34]. Concurrency bugs usually occur due to thread synchronization issues and are extremely hard to diagnose [35]. A supervisor, based on automatically extracted software models, can constrain software behavior to prevent runtime failures. Theory and tools for deadlock avoidance by supervision in concurrent programming are still being developed, but experiments show seamless elimination of common deadlock occurrences, while merely doubling compile time [36,37].

Supervisory controller synthesis has proven to be a valuable and industrially-viable technique in the situations when the control requirements change often, the system evolves during the product development process, or there is undesired complex emergent behavior. However, despite recent success, there are still many challenges that need to be address to enable a wide-spread use of this technique. As with other discrete-event based techniques, like formal verification, supervisory controller synthesis suffers from the state-explosion problem. This is somewhat mitigated in state-of-the-art tools by employing efficient data structures, like binary decision diagrams [32].

However, further development is needed in decentralized, modular, and hierarchical supervisory control, that employ the structure of the system to synthesis a set of concurrent and/or communicating local supervisors responsible for an abstracted part of the system that are possibly coordinated by a higher level (global) supervisory coordinator. Here, we believe that our process-theoretic approach can provide for a novel perspective and offer improved techniques, as existing trace-based approaches often struggle with nondeterminism that inevitably occurs during the abstraction of the local behavior.

Furthermore, we are often impelled by the industry to also ensure that the supervised system has all of the prescribed functionalities and satisfies given performance specifications. It is not trivial to include such requirements in the synthesis process due to high complexity issues. To this end, the process-theoretic approach can provide for suitable abstractions that allow us to decouple the synthesis of a safe and nonblocking supervisor from the extraction of optimal schedulers, enabling us to solve these tasks using the most efficient dedicated tools [28].

References

[1] M. Hinchey, J. Bowen, Applications of Formal Methods, International Series in Computer Science, Prentice-Hall, 1995.
[2] T. Mertke, T. Menzel, Methods and tools to the verification of safety-related control software, in: Proceedings of SMC 2000, vol. 4, 2000, pp. 2455–2457.
[3] R.T.C.A. Inc., EUROCAE, DO-178B: Software considerations in airborne systems and equipments certification, 1992.
[4] UK Ministry of Defence, Defence standard 00-55 – The procurement of safety critical software in defence equipment, 1997.
[5] J. Woodcock, P.G. Larsen, J. Bicarregui, J. Fitzgerald, Formal methods: Practice and experience, ACM Comput. Surv. 41 (4) (2009) 1–36.
[6] F. Iwu, A. Galloway, J. McDermid, I. Toyn, Integrating safety and formal analyses using UML and PFS, Reliab. Eng. Syst. Saf. 92 (2) (2007) 156–170.
[7] T. Kim, D. Stringer-Calvert, S. Cha, Formal verification of functional properties of a SCR-style software requirements specification using PVS, Reliab. Eng. Syst. Saf. 87 (3) (2005) 351–363.
[8] S. Cha, H. Son, J. Yoo, E. Jee, P.H. Seong, Systematic evaluation of fault trees using real-time model checker UPPAAL, Reliab. Eng. Syst. Saf. 82 (1) (2003) 11–20.
[9] J. Lahtinen, J. Valkonen, K. Bjorkman, J. Frits, I. Niemela, K. Heljanko, Model checking of safety-critical software in the nuclear engineering domain, Reliab. Eng. Syst. Saf. 105 (2012) 104–113.
[10] N. Leveson, The challenge of building process-control software, IEEE Softw. 7 (6) (1990) 55–62.
[11] S. Anderson, M. Felici, Safety, reliability and security of industrial computer systems, Reliab. Eng. Syst. Saf. 81 (3) (2003) 235–238.
[12] R.R.H. Schiffelers, R.J.M. Theunissen, D.A.v. Beek, J.E. Rooda, Model-based engineering of supervisory controllers using CIF, Electron. Comm. EASST 21 (2009) 1–10.
[13] J. Markovski, D.A. van Beek, R.J.M. Theunissen, K.G.M. Jacobs, J.E. Rooda, A state-based framework for supervisory control synthesis and verification, in: Proceedings of CDC 2010, IEEE, 2010, pp. 3481–3486.
[14] J.C.M. Baeten, J.M. van de Mortel-Fronczak, J.E. Rooda, Integration of supervisory control synthesis in model-based systems engineering, in: Proceedings of ETAI/COSY 2011, IEEE, 2011, pp. 167–178.
[15] C. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers, 2004.
[16] P.J. Ramadge, W.M. Wonham, Supervisory control of a class of discrete-event processes, SIAM J. Control Optim. 25 (1) (1987) 206–230.
[17] P. Tabuada, G.J. Pappas, Linear time logic control of discrete-time linear systems, IEEE Trans. Autom. Control 51 (12) (2006) 1862–1877.
[18] N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive(1) designs, in: VMCAI 2006, in: LNCS, vol. 3855, Springer, 2006, pp. 364–380.
[19] A. Morgenstern, K. Schneider, A LTL fragment for GR(1)-synthesis, in: Proceedings of iWIGP, in: EPTCS, vol. 50, Open Publishing Association, 2011, pp. 33–45.
[20] F. Cassez, A. David, E. Fleury, K.G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: Proceedings of CONCUR 2005, in: Lecture Notes in Computer Science, vol. 3653, Springer, 2005, pp. 66–80.
[21] A. Overkamp, Supervisory control using failure semantics and partial specifications, IEEE Trans. Autom. Control 42 (4) (1997) 498–510.
[22] M. Heymann, F. Lin, Discrete-event control of nondeterministic systems, IEEE Trans. Autom. Control 43 (1) (1998) 3–17.
[23] G. Barrett, S. Lafortune, Bisimulation, the supervisory control problem and strong model matching for finite state machines, Discrete Event Dyn. Syst. 8 (4) (1998) 377–429.
[24] J.J.M.M. Rutten, Coalgebra, concurrency, and control, in: Proceedings of WODES 2000, Kluwer, 2000, pp. 31–38.
[25] C. Zhou, R. Kumar, S. Jiang, Control of nondeterministic discrete-event systems for bisimulation equivalence, IEEE Trans. Autom. Control 51 (5) (2006) 754–765.
[26] J.C.M. Baeten, D.A. van Beek, B. Luttik, J. Markovski, J.E. Rooda, A process-theoretic approach to supervisory control theory, in: Proceedings of ACC 2011, IEEE, 2011, pp. 4496–4501.
[27] J. Baeten, D. van Beek, A. van Hulst, J. Markovski, A process algebra for supervisory coordination, in: Proceedings of PACO 2011, in: EPTCS, vol. 60, Open Publishing Association, 2011, pp. 36–55.
[28] J. Markovski, M. Reniers, Verifying performance of supervised plants, in: Proceedings ACSD 2012, IEEE, 2012, pp. 52–61.
[29] R.J.M. Theunissen, R.R.H. Schiffelers, D.A. van Beek, J.R. Rooda, Supervisory control synthesis for a patient support system, in: Proceedings of ECC 2009, EUCA, 2009, pp. 1–6.
[30] J. Markovski, K.G.M. Jacobs, D.A. van Beek, L.J.A.M. Somers, J.E. Rooda, Coordination of resources using generalized state-based requirements, in: Proceedings of WODES 2010, IFAC, 2010, pp. 300–305.
[31] S.T.J. Forschelen, J.M. Mortel-Fronczak, R. Su, J.E. Rooda, Application of supervisory control theory to theme park vehicles, Discrete Event Dyn. Syst. 4 (2012) 511–540.
[32] C. Ma, W.M. Wonham, Nonblocking Supervisory Control of State Tree Structures, Lecture Notes in Control and Information Sciences, vol. 317, Springer, 2005.
[33] G.S. Waliaa, J.C. Carverb, A systematic literature review to identify and classify software requirement errors, Inf. Softw. Technol. 51 (7) (2009) 1087–1109, http://dx.doi.org/10.1016/j.infsof.2009.01.004.
[34] Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, S. Mahlke, The theory of deadlock avoidance via discrete control, in: Proceedings of POPL 2009, ACM, 2009, pp. 252–263.

[35] H. Sutter, J. Larus, Software and the concurrency revolution, Queue 3 (7) (2005) 54–62.
[36] T. Kelly, Y. Wang, S. Lafortune, S. Mahlke, Eliminating concurrency bugs with control engineering, Computer 42 (12) (2009) 52–60.
[37] Y. Wang, H.K. Cho, H. Liao, A. Nazeem, T.P. Kelly, S. Lafortune, S. Mahlke, S.A. Reveliotis, Supervisory control of software execution for failure avoidance: Experience from the Gadara project, in: Proceedings of WODES 2010, IFAC, 2010, pp. 269–276.