Emilia Cioroica, Fraunhofer IESE
Karsten Albers, INCHRON AG
Wolfgang Boehm, Technical University of Munich
Florian Pudlitz, Technische Universität Berlin
Christian Granrath, RWTH Aachen University
Roland Rosen, Siemens AG
Jan Christoph Wehrstedt, Siemens AG

12

Development and Evaluation of Collaborative Embedded Systems using Simulation

Embedded systems are increasingly equipped with open interfaces that enable communication and collaboration with other embedded systems, thus forming collaborative embedded systems (CESs). This new class of embedded systems, capable of collaborating with each other, is planned at design time and forms collaborative system groups (CSGs) at runtime. When they are part of a collaboration, systems can negotiate tactical goals, with the aim of achieving higher level strategic goals that cannot be achieved otherwise. The design and operation of CESs face specific challenges, such as operation in an open context that dynamically changes in ways that cannot be predicted at design time, collaborations with systems that dynamically change their behavior during runtime, and much more. In this new perspective, simulation techniques are crucially important to support testing and evaluation in unknown environments. In this chapter, we present a set of challenges that the design, testing, and operation of CESs face, and we provide an overview of simulation methods that address those specific challenges.

12.1 Introduction

Modeling and simulation are established scientific and industrial methods to support system designers, system architects, engineers, and operators of several disciplines in their work during the system life cycle. Simulation methods can be used to address the specific challenges that arise with the development and operation of collaborative embedded systems (CESs). In particular, the evaluation of collaborative system behavior in multiple, complex contexts, most of them unknown at design time, can benefit from simulation. In this chapter, after a short motivation, we exemplify scenarios where simulation methods can support the design and the operation of CESs and we summarize specific simulation challenges. We then describe some core simulation techniques that form the basis for further enhancements addressed in the individual chapters of this book.

12.1.1 Motivation

Simulation is a technique that supports the *overall design, evaluation, and trustworthy operation* of systems in general. CESs are a special class of embedded systems that, although individually designed and developed, can form collaborations to achieve collaborative goals during runtime. This new class of systems faces specific design and development challenges (cf. Chapter 3) that can be addressed with the use of simulation methods.

At *design time*, a suitable simulation allows verification and exploration of the system behavior and the required architecture based on a virtual integration. At *runtime*, when systems operate in open contexts, interact with unknown systems, or activate new¹ system functions, the aspect of *trust* becomes of crucial importance. Using later research and technology advancements, we foresee the possibility of computing trust scores of CESs directly at runtime based on the evaluation results of system behavior in multiple simulated scenarios. The core simulation techniques presented in this chapter form the basis for enhanced testing and evaluation techniques.

¹ “New functions” are functions that have not been enabled before in the current internal system configuration.

12.1.2 Benefits of Using Simulation

Regardless of the domain, the use of simulation methods for behavioral evaluation of systems and system components has multiple benefits.

For a concrete scenario of complex interactions, simulation methods are more exploratory than analytical methods. The effectiveness of the exploration is achieved through the coupling of detailed simulation models, while the efficiency of the exploration is achieved by exercising a system or system group behavior in a multitude of scenarios, including scenarios that contain failures.

Simulation to support effectiveness of exploration

Through the collaboration of CESs, collaborative system groups (CSGs) that did not exist before are formed dynamically at runtime. Moreover, the exact configuration of those CSGs is not known at design time. In such situations, when systems operate in groups that never existed before, there is insufficient knowledge about the collaborative behavior and its effects. In this case, simulation can help to discover the effects of different function interactions.

Simulation to evaluate the function interaction

As a third benefit, the use of closed-loop simulation (X-in-the-loop simulation) is a suitable approach for testing embedded systems (e.g., control units of collaborative assistant systems). The independence of the simulated test environment from the implementation and realization of the embedded system (system under test) generates advantages, such as reusability of the simulations and cost savings in system testing. One example is the testing of different control units—for which the simulation environment can be reused without major adaptations—independently of the implementation and realization concept of the control unit. Only the interfaces of realized functionality of the system under test have to be the same to enable coupling of the simulation and testing environment.

Closed loop simulation

A fourth major benefit is that the risk for the system user (e.g., car passenger) can be reduced by using simulations during the system testing process by virtual evaluation. The test execution in virtual environments enables discovery of harmful behavior in a virtual world, where only virtual and not real entities are harmed. Real hazards can thus be avoided. In addition, the risk during the operation of collaborative systems can be reduced by using predictive risk assessment by means of simulation.

Risk reduction

Additionally, the use of simulations for testing at system design time can be used to make tests virtual, with an associated reduction in hardware and prototypes. In particular, the costs for the production of these real components can be reduced. In addition, making tests virtual leads to early error detection and correction and thus to a

Virtualization of tests

*Reduction of
development time
and costs*

further reduction in development costs. This is especially useful as the exact configuration of CSGs is not known at design time. Here, simulation gives the opportunity to simulate sets of possible (most likely) scenarios.

Furthermore, the independence of simulation models that reflect the behavior of real components results in efficient development, because in some use cases, simulations are not bound to real-time conditions. Therefore, they can be executed much faster than in real time and thus be used to reduce development time. It is also easier to explore many more scenarios and variations of scenarios to gain a better overview and trust in the systems.

As a seventh benefit, the use of simulation environments for testing embedded systems is especially independent of external influences of the environment and ensures that tests can be reproduced. This allows efficient tracking and resolution of problems exposed by the simulation and reproduction of the absence of the problems in the updated system configuration.

The last benefit is that the internal behavior of the simulated systems and their visualization are exposed in a broad way. The traceability of the execution of a real system is limited due to hardware and time restrictions. In the simulation, it is easier to log relevant internal system execution and therefore to identify the causes of problems and unexpected behavior.

In the context of developing and evaluating CESs, the use and benefit of simulation—as described above—lie mainly in the first phases of the entire life cycle. In addition, simulation is also used during operation and service—that is, during the runtime of the system. Thus, simulation represents a methodology that can be used seamlessly across all life cycle phases. Accordingly, there are different challenges for simulation as a development methodology and as a validation technique.

12.2 Challenges in Simulating Collaborative Embedded Systems

Even though there are multiple benefits from using simulation, the aspect of simulation for CESs and CSGs poses particular challenges. In this section, we describe the design time and runtime challenges.

12.2.1 Design Time Challenges

To support the use of simulation during the design of collaborative systems, as presented in Chapter 3, multiple challenges must be addressed, as detailed in the following.

One challenge is the *evaluation of function interaction at design time*, because in a simulation of CESs, functions of multiple embedded systems, developed independently, must be integrated to allow evaluation of the resulting system. This is necessary to discover and fix unwanted side effects before the systems are deployed in the real world. Also, the other relevant aspects for the simulation scenario, such as the context or the dynamic behavior of the systems, must be covered. To support this activity, the *integration of different models and tools* is also important. Development of collaborative system behavior relies on simulating models of different embedded systems that are often developed with different tools. Furthermore, the *integration of different simulation models*, sometimes at different levels of detail, represents an important design engineering challenge. This is because the design of CESs relies on the evaluation of collaborative system behavior that can be expressed at different levels of abstraction. Another challenge is the *integration of different aspects of the simulation scenario*. The comprehensive simulation of collaboration scenarios must cover several aspects to achieve a broad coverage of scenarios. Examples are the context of the CSG, the execution platform of these systems and the system group, including the functional behavior, the timing behavior, and the physical behavior of the systems and the system group. The different aspects can require dedicated models and must therefore be covered by specialized simulation tools. For a comprehensive simulation of the whole scenario, these models and tools must interact with each other and must be integrated via a co-simulation platform.

The use of simulation methods pursues specific strategic goals as well. One of these methods is the *virtual functional test*, which uses simulation to test a certain collaboration functionality or a certain functionality of one system in the collaborating context. The models of the other parts (systems, context, etc.) must include only those details relevant for the functionality being tested.

Virtual functional test

Another purpose of the simulation is the *virtual integration test*. Here, simulation tests the correct collaboration of the different systems or parts of the systems in a virtual environment. The exact structure of the CSG may not be available at design time and can be subject to dynamic changes. Simulation can test multiple scenarios for this structure for a multitude of situations. An early application of

Virtual integration test

such tests in the design process, before the different systems are fully designed and implemented, will allow early detection of potential problems and hazards for the collaboration behavior.

*Design-space
exploration*

One strategic goal for the application of simulation, especially in early design phases, is to support a *design-space exploration*. The possibility to support the evaluation of a lot of design alternatives and to identify hazards and failures in the different simulation models allows a strategic evolutionary search for a system variant that fulfills the desired goals and requirements.

*Fulfillment of
requirements*

The *determination of fulfilled requirements* allows the simulations to serve as automation tools for test cases. The results must then be linked to the requirements to determine the coverage. Besides the degree of coverage, additional system behavior can be investigated in relation to the requirements. Due to the great complexity of collaborative systems, automated algorithms must be increasingly used. In Section 12.3, we present a possible approach to help developers and testers meet this challenge.

12.2.2 Runtime Challenges

Even though properly tested during design time, CESs face multiple challenges at runtime and the simulation techniques deployed at runtime face particular challenges as well. In this subsection, we list the challenges of CESs and CSGs as introduced in Chapter 2. We then detail the challenges of using simulation to solve these runtime challenges.

Open context

One particular challenge CESs face at runtime is *operation in open contexts*. The external context may change in unpredictable ways during the runtime operation of CESs. In particular, the environment changes and the context of collaboration may change as well. For example, in the automotive domain, a vehicle that is part of a platoon may need to adapt its behavior when the platoon has to reduce the speed due to high traffic. If the vehicle has a strong goal of reaching the target destination at a specific time, it may decide to leave the platoon that is driving at a lower speed and select another route to its destination. For the remaining vehicles within the platoon, the operational context has changed because the vehicle is now no longer part of the platoon and instead, becomes part of the operational context.

The operational context of a CSG may change dynamically as well, either because a CES joins the group or because the CSG has to operate in an environment that was not foreseen at design time. The CSG has

to adapt its behavior in order to cope with the new environmental conditions. For example, a vehicle under the control of a system function in charge of maintaining a certain speed limit within a platoon has difficulty maintaining the speed after it starts raining.

When CESs form at runtime, the runtime activation of system functions poses additional challenges. When the behavior of CESs is coordinated by the collaboration functions that negotiate the goals of the systems and activate system functions, multiple challenges arise when these system functions are activated for the first time. One example is scheduling: the timing behavior of system functions activated for the first time can influence the scheduling behavior of (a) the interacting system functions, (b) the collaboration functions, and (c) of the whole system.

In this case, the functional interaction must be evaluated because when *system functions are activated for the first time*, the way in which they interact with other system functions in specific situations can be faulty.

Moreover, *changing goals at runtime* can also have consequences on the CSG or the CESs. In order to form a valid system group, CESs and/or the CSG may need to change their goals at runtime dynamically, which may obviously have significant impact on the system behavior.

*Runtime evaluation of
changing goals*

The overall *dynamic change of internal structures within a CSG* is impossible to foresee at design time. When a CES leaves a CSG, the roles of the remaining participants and their operational context may change as well. The same happens when a new vehicle joins the platoon as a platoon participant that later on may take the role of platoon leader. In turn, this leads to a *dynamic change of system borders of a CSG*, which may change the overall functionality of the CSG. For example, a vehicle ahead of the platoon is considered a context object that influences the speed adjustments of the approaching platoon. If the vehicle in front of the platoon decided to join the platoon, then the borders of the initial platoon would be extended.

*Runtime evaluation of
changes in the internal
structure*

Addressing the challenges mentioned above by using simulation may even require using simulation at runtime, which, in turn, puts further requirements on the simulation method.

*Using simulation at
runtime*

Firstly, when simulation is used to control the behavior of safety-critical systems, the *real-time deadlines* must be achieved. When system behavior is evaluated at runtime, in a simulated environment, then the simulation must deliver the results on time. This is necessary in order to give the system the chance of executing a safe failover

*Meeting real-time
deadlines*

behavior if the virtual evaluation discovers hazardous behavior of the system under operation.

Secondly, predictive evaluation of system behavior is possible only by achieving efficient simulation models. When system behavior is evaluated at runtime, in a simulated environment, it must execute faster than the wall clock. This imposes a high degree of efficiency on the simulation models that are executed. For example, it may not be feasible to execute detailed simulation models as parts of the interacting platform because this may take too much time. Instead of executing the detailed models, abstractions of the system behavior can be executed. These abstractions must be directed towards the scope of the evaluation. If scheduling behavior needs runtime evaluation in a simulated environment, then the parts of the platform that influence or are influenced by the scheduling will be executed.

*Enabling model
abstraction to achieve
efficiency*

However, in order to have accurate evaluation, the efficiency of simulation must balance with the effectiveness of simulation models. In order to perform a trustworthy system evaluation in a simulation environment during runtime, the models must accurately reflect the parts of the system under evaluation. However, because simulation also needs to be efficient, effective simulation can be achieved by using the abstraction models (for efficiency reasons) directed towards the scope of the evaluation. This in turn requires extensive effort during the design time of the system to create accurate models that reflect selected parts (abstraction) of the internal system architecture. For example, to enable evaluation of scheduling at runtime, systems engineers must design the meaningful simulation models of the platform that will be executed during scheduling analysis.

12.3 Simulation Methods

Simulation is a universal solution approach and is based on the application and use of a few basic concepts from numerical mathematics. In our case, simulation models are implemented in software and use numerical algorithms for calculation. We speak of time-discrete, discrete-event, or continuous simulation (continuous time) depending on the mathematical concepts used, which characterize the different handling of time behavior. Simulation tools usually realize a combined strategy. The fact that simulation covers several disciplines, combines different elements of a system, or addresses the system and its context, leads to approaches for a cooperation of different simulations, also called co-simulation. From

a practical point of view, data and result management are important for supporting the simulation activities.

In the area of testing software functions, the three approaches Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), and Hardware-in-the-Loop (HIL) are relevant [VDI 3693 2016]. MIL simulation describes the testing of software algorithms implemented prototypically during the engineering phase. These algorithms are implemented using a *simulation models language*, mostly in the same simulation tool that is also used to simulate the physical system (understood here as the dynamic behavior with its multidisciplinary functions) itself. The SIL simulation describes a subsequent step. The software is realized in the original programming or automation language and is executed on emulated hardware and coupled with a simulation model of the physical system. The third step is a HIL simulation. Here, the program (or automation) code compiled or interpreted and executed on the target hardware is tested against the simulation of the physical system.

Simulation of technical systems usually consists of three steps: model generation (including data collection), the execution of simulation models, and the use of the results for a specific purpose. In the following, we describe the methodology of simulation for these three process steps.

In general, the data collection and generation of the models take a lot of effort and time. For virtual commissioning, there are statements that up to two-thirds of the total time is spent on these activities [Meyer et al. 2018]. As a consequence, especially for CESs and CSGs in partially unknown contexts, efficient methods for setting up the model must be provided. Integrating the model generation directly into the development process in order to generate up-to-date models at any time is a good approach, as shown in Chapter 6.

*Supporting model
creation*

The most common concept for seamless integration of all information relevant in the entire life cycle of a product is product lifecycle management (PLM). It integrates all data, models, processes, and further business information and forms a backbone for companies and their value chains. PLM systems are, therefore, an important source for the creation of simulation models.

With the technical vision of a digital twins approach, the importance of different kinds of models is increased. Digital twins are abstract simulation models of processes within a system fed with real-time data. For more information on supporting the creation of digital twins for CESs, see Chapter 14. Semantic technologies are used to realize the interconnectedness of all information and to guarantee the

openness of the approach to add further artifacts at any time [Rosen et al. 2019]. These semantic connections, frequently realized by knowledge graphs, can be used in future to generate executable simulation models that are up to date with all available information more efficiently.

*Enabling model
execution*

Furthermore, existing models must be combined to form an overall model of different aspects of the system and context. This requires an exchange of models between different tools, which can be solved via co-simulation [Gomes et al. 2017]. The FMI standard [FMI 2019] describes two approaches towards co-simulation. With *model exchange*, only those models that can be solved with one single solver are combined to form an overall mathematical simulation model, whereas *FMI for co-simulation* uses units, consisting of models, solvers, etc. that are orchestrated by a master. On the one hand, this master must match the exchange variables described in the interfaces. On the other hand, it must orchestrate the different time schemes of the different simulators from discrete-event through time-discrete up to continuous simulation [Smirnov et al. 2018]. For efficient simulation of CSGs, the simulation chains must therefore be set up and modified quickly and efficiently as they can change quite often depending on the situation.

In order to set up an integrated development and modeling approach, two aspects must be covered: firstly, different methods must be assembled into an integrated methodology; and secondly, interoperability and integration between different tools must be established in order to set up an integrated tool chain (see Chapter 17). A special focus of co-simulation lies in HIL simulation, which uses real control hardware. The remaining simulation models, with their inherent simulation time, must be executed faster than real-world time to ensure that the results are always available at the synchronization time points with the physical HIL system. Thus, both, the slowest model as well as the orchestration process, must be executed faster than real-world time.

*Developing the use of
the results*

One key goal of simulation is validation and testing of the system behavior. This requires the definition of test cases, the setup of the simulation model, execution of the test cases, and finally, the evaluation of the test. For context-aware CESs and CSGs in particular, this may be a highly complex task with exponentially increasing combinations. Finally, the test results must be compared with the requirements. In Chapter 15, we therefore develop exhaustive testing methods to cope with these challenges.

One way to support the tester is to mark system-relevant information in the requirements and link it to simulation events. A markup language can be used to mark software functions and context conditions within a document. After important text passages in the requirements have been marked, they can be extracted automatically. When the extraction process is completed, the information is linked to the specific signals of the system. This results in a mapping table. Since many simulators, models, and interfaces are used in the simulation of CESs, a central point is created to combine them. In the simulation phase, all signals of the function under test are recorded and stored in log data. These log data contain all signal names and their values for each simulation step. Once the simulation run is complete, the log data can be processed further and linked to the original requirements using the mapping table from the previous phase. This allows the marked text phrases in the requirements to be evaluated and displayed to the user.

Simulation methods are increasingly integrated into the design and development process and used in all phases of the system life cycle [GMA FA 6.11 2020]. Beyond development, validation, and testing, simulation is used during operation with an increasing benefit [Schlegner et al. 2017]. Specific applications include simulations in parallel to operation in order to monitor, predict, and forecast the behavior of the CESs. This means that simulation models must be updated regarding the current state of the systems collaborating in a CSG [Rosen et al. 2019]. Chapter 3 introduces a flexible architecture for the integration of simulation into the systems architecture to support the decision of the system or the operator.

Integration into process

For complex scenarios, the simulation has to cover not only the functional behavior of a single system, but also the combined behavior of the CSG and all relevant aspects, including, for example, the resulting collaboration behavior, the context of the collaborative system, the timing of the systems, and the communication between the systems and with the context. The collaboration functions result from the interaction between the functions of the different systems. All these aspects must be addressed by simulation as early as possible in the design process. It may not be sufficient to test them in a HIL simulation when the implementation of the system has already widely progressed. The MIL and SIL simulations must also address those aspects.

12.4 Application

The methods described above have several applications. First of all, they support development, testing, and virtual integration, especially in early phases of the system design. They also support the development of extended simulation methods such as the ones used for runtime evaluation of system trustworthiness, as presented in Chapter 10; they support the generation of simulation models based on a step-by-step approach, as presented in Chapter 6; and they support the operator during system operation, as presented in Chapter 3. Furthermore, they support system evaluation in real-world scenarios.

Simulation methods for development, testing, and virtual integration

During the design of CESs in particular, simulation methods can help to check the current state of development, verify the correctness and completeness of the current design, and explore the applicability of the next steps and extensions. For collaborative systems, virtual integration of different systems is a special challenge, especially in early and incomplete stages of development. The purpose is to explore the collaborative behavior as early as possible, detect possible hazards and failures when they are much easier to change, and adapt the design of the systems for the solution to these hazards and failures.

Simulating the collaborative behavior in the early stages of development—especially for applications like autonomous driving—should include all relevant aspects of the underlying scenarios, especially context and physical system behavior. Co-simulation approaches can address the challenges involved in such a comprehensive simulation. Chapter 13 provides more details on the possibilities and tools for realizing such simulation approaches.

Simulation methods as a basis for extension

Building trust into collaborative embedded systems requires a sustained evaluation and testing effort that spans from design time to runtime. As detailed in the sections above, simulation is an important technique that enables system and software testing at design time and behavior evaluation during runtime. Within CrEST, as presented in Chapter 10, an extension of existing simulation methods has been realized. These methods either address runtime challenges at design time or enable runtime evaluation of system behavior.

Simulation methods for runtime evaluation

Addressing runtime challenges at design time is enabled by extending the co-simulation method described in this chapter towards integrating the real world (in which collaboration functions and system functions execute on real hardware) with the virtual world (formed by purely virtual entities). This allows the runtime

activation of system functions, for example, to be validated in an extended set of scenarios that are easier and cheaper to explore within a virtual environment.

Building on the challenges and methods described in this chapter, simulation techniques deployable at runtime have been developed. Coupled with monitoring components, simulation can be used for runtime prediction of system behavior emerging from the runtime activation of system functions. When simulation platforms are deployed on CESs, the functional and timing interaction of a collaboration function with system functions and the functional and timing interactions between system functions can be predicted at runtime. For details on how the simulated prediction is performed, see Chapter 10 of this book.

12.5 Conclusion

Simulation methods support the development of CESs, verification and validation of their continuous development, from the conceptual phase when abstract behavioral methods can be coupled through co-simulation and verification of system behavior after detailed models are integrated, up to the final testing of systems before deployment. We have analyzed the benefits and challenges of CESs and of simulation methods that support their development and testing. We have set the basis for future extensions beyond the current state of the art and practice.

In order to realize these technological visions, it is important to consider the economic benefits. This means that the effort and ultimately the cost of deployment must not exceed the benefits. One approach will be a step-by-step realization. This will ensure that advanced simulation methods will be a success factor for validation and testing of CESs.

12.6 Literature

[Alexander and Maiden 2004] I. Alexander, N. Maiden (Eds.): *Scenarios, Stories, Use Cases – Through the Systems Development Life-Cycle*. Wiley, Chichester, 2004.

[Allmann et al. 2005] C. Allmann, C. Denger, T. Olsson: *Analysis of Requirements-Based Test Case Creation Techniques*. IESE-Report No. 046.05/E, Version 1.0, June 2005. Fraunhofer-Institute for Experimental Software Engineering IESE, Kaiserslautern, 2005.

[FMI 2019] Functional Mock-up Interface (FMI) Specification 2.0.1, <https://fmi-standard.org/>, accessed 03/01/2020.

- [GMA FA 6.11 2020] R. Rosen, J. Jäkel, M. Barth: VDI Statusreport 2020: Simulation und digitaler Zwilling im Anlagenlebenszyklus. VDI/VDE, 2020 (available in German only).
- [Gomes et al. 2017] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe: Co-Simulation: State of the Art. arXiv preprint arXiv:1702.00686, 2017.
- [Meyer et al. 2018] T. Meyer, S. Munske, S. Weyer, V. Brandstetter, J. C. Wehrstedt, M. Keinan: Classification of Application Scenarios for a Virtual Commissioning of CPS-Based Production Plants into the Reference Architecture RAMI 4.0. In: Proceedings of the VDI AUTOMATION-18: Seamless Convergence of Automation & IT, 2018.
- [Rosen et al. 2019] R. Rosen, J. Fischer, S. Boschert: Next Generation Digital Twin: An Ecosystem for Mechatronic Systems? In: 8th IFAC Symposium on Mechatronic Systems MECHATRONICS 2019: Vienna, Austria, 2019.
- [Schegner et al. 2017] L. Schegner, S. Hensel, J. Wehrstedt, R. Rosen, L. Urbas: Architekturentwurf für simulationsbasierte Assistenzsysteme in prozesstechnischen Anlagen. Tagungsband Automation 2017, Baden-Baden 2017 (available in German only).
- [Smirnov et al. 2018] D. Smirnov, T. Schenk, J. C. Wehrstedt: Hierarchical Simulation of Production Systems. In: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE 2018, pp. 875-880.
- [VDI 3693 2016] VDI/VDE 3693 Blatt 1:2016-08. Virtual commissioning - Model types and glossary. Berlin: Beuth Verlag.
- [Zheng et al. 2014] Y. Zheng, S. E. Li, J. Wang, K. Li: Influence of Information Flow Topology on Closed-Loop Stability of Vehicle Platoon with Rigid Formation. In: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), IEEE, pp. 2094-2100.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.