Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 4, October, 2015, Pages 50-65
Published online by RTU Press, https://csimq-journals.rtu.lv
http://dx.doi.org/10.7250/csimq.2015-4.04
ISSN: 2255-9922 online

Early Validation of Automation Plant Control Software using Simulation Based on Assumption Modeling and Validation Use Cases

Veronika Brandstetter[1], Andreas Froese[2], Bastian Tenbergen[2,3*], Andreas Vogelsang[4], Jan Christoph Wehrstedt[1], Thorsten Weyer[2]

[1] Siemens AG, Corporate Technology, Germany
[2] paluno – The Ruhr Institute for Software Technology, Univ. of Duisburg-Essen, Germany
[3] Department of Computer Science, State University of New York, Oswego, NY, USA
[4] Technische Universität München, Germany

{veronika.brandstetter|janchristoph.wehrstedt}@siemens.com,
{andreas.froese| thorsten.weyer}@paluno.uni-due.de, bastian.tenbergen@oswego.edu,
vogelsan@in.tum.de

Abstract. In automation plants, technical processes must be conducted in a way that products, substances, or services are produced reliably, with sufficient quality and with minimal strain on resources. A key driver in conducting these processes is the automation plant's control software, which controls the technical plant components and thereby affects the physical, chemical, and mechanical processes that take place in automation plants. To this end, the control software of an automation plant must adhere to strict process requirements arising from the technical processes, and from the physical plant design. Currently, the validation of the control software often starts late in the engineering process in many cases – once the automation plant is almost completely constructed. However, as widely acknowledged, the later the control software of the automation plant is validated, the higher the effort for correcting revealed defects is, which can lead to serious budget overruns and project delays. In this article we propose an approach that allows the early validation of automation control software against the technical plant processes and assumptions about the physical plant design by means of simulation. We demonstrate the application of our approach on the example of an actual plant project from the automation industry and present it's technical implementation.
Keywords: Automation technology, process plants, context modeling, simulation, validation, executable requirements, assumptions.

1 Introduction

The development of automation plants (e.g., processing facilities in the chemical industry, production facilities in factories, or baggage routing facilities at airports) is a complex planning and engineering task. Typically, such automation plants are highly customized and the entire construction process from the first idea to commissioning takes years, involving many different disciplines like process engineering, physical plant design, mechanics, electronics, and software engineering [1].

In our experience, the development of control software for automation plants becomes more and more challenging. The overall goal of the automation control software is the control of the technical plant processes performed within the automation plant in order to process or produce

products, substances, or services of a sufficient quality with the lowest demand on resources, such as time, energy, and material input [2]. For this purpose, the automation control software acts as the application software of the automation plant and calculates functions, solves equations, and controls the various technical devices of the automation plant (e.g., valves, containers, or conveyor belts). Moreover, physical properties of the involved materials (e.g., maximum pipe pressures, material cooling cycles, or friction) constrain the technical processes considerably. In consequence, the desired behavior of the automation control software is strongly influenced by strict process requirements, which arise due to the production method, involved materials, and plant components. Validating whether the automation control software satisfies the process requirements of the entire plant is usually conducted at a late stage in the plant development process, when the technical processes and the physical plant design are already defined or even constructed. Nevertheless, it is widely acknowledged that the later the control software of the automation plant is validated, the higher the effort for correcting revealed defects will be. This can lead to serious budget overruns, project delays, or to automation plants that do not fulfil their operational purpose.

In [3] we proposed an approach that fosters the *early* validation of automation control software against the technical plant process, based on specified requirements of the control software and assumptions about the physical plant design. *Early* in the context of this research means that validation takes place as soon as information about the technical process and the involved plant components is available, i.e., when conceptualization of the plant is complete, but the plant has not been built yet. Our approach aims at the identification of defects in the requirements (in the sense of incorrect or incompletely specified requirements) for the automation control software. The key principle of the approach is to use simulation during early stages of plant development process to assess the impact of the specified requirements for the automation control software on the technical plant process. This article is an extension of previous work [3] and includes additional details regarding automation industry, the desalination plant example, and a tooling implementation of our approach in Section 3.3.3. Furthermore, this article explains the nature and application of context models in Section 3.1 in greater detail and provides a meta-model, which integrates static-structural, functional, and behavioral context model perspectives. In addition, this article describes the co-simulation of plant processes in detail and compares our approach to established techniques from the literature.

The approach is primarily aimed at validating requirements for control software of automation plants. Yet, we expect that our approach can be widely used to validate application software for technical systems with complex technical and physical constraints. This applies, in particular, to systems in which application software controls immutable technical processes and in which valid assumptions about the physical design of the system can be made early in the development process.

The remainder of the article is structured as follows: Section 2 introduces the running example of a seawater desalination plant, which is used in this article to demonstrate the core concepts of our approach. Section 3 describes our approach for the early validation of automation control software in the development of automation plants. Section 4 reports on the findings from the evaluation of our approach in an industrial environment. Section 5 gives an overview on the related work. Finally, Section 6 concludes the article.

2 Running example

We illustrate our approach by means of a seawater desalination plant. Desalination plants are used to remove salts from seawater for the purpose of producing drinking water. Desalination is achieved using reverse osmosis, i.e., a filtration method, which requires water to be pumped through membranes at high pressure. An overview of a common technical plant architecture of such a plant is given in Figure 1. In our simplified running example, seawater is collected

through a system of four beach wells, which are drilled into the seashore. From the beach wells, salty seawater is pumped through pipelines to a seawater tank, where it is collected and pretreated with various chemicals for stabilization and biochemical cleansing before desalination can take place.

In the remainder of the aricle we will focus on the beach wells. A beach well has the following general tasks:

- Collect seawater through subsurface intakes, drilled into the seashore
- Filter seawater through natural sand layers
- Pump water from the subsurface intake collection tank to the seawater collection tank
- Adjust the flow of seawater pumped into the seawater tank according to some constraints

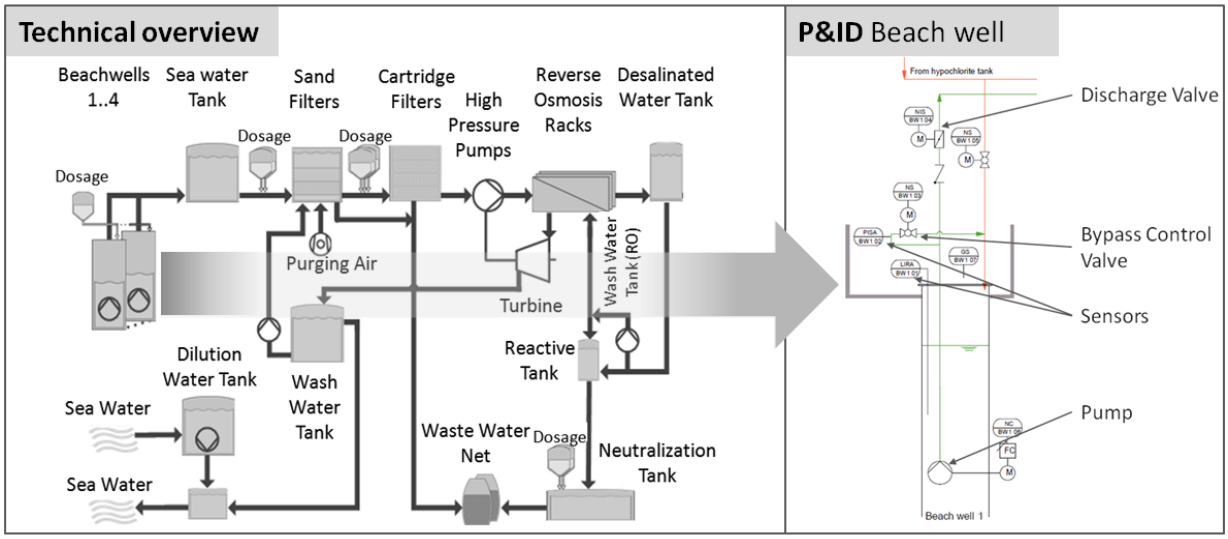

Figure 1. Technical plant architecture of a typical seawater desalination plant and one beach well

In the automation industry, technical plant architectures are designed and documented by piping and installation diagrams (P&ID). Figure 1 shows the P&ID for one of the four beach wells with its main components. Each beach well is equipped with a pump to advance collected water, a discharge valve, through which water is advanced to the seawater tank, and a bypass control valve, which can be used to adjust the flow rate into the seawater tank. These beach well components are controlled by the beach well software. The beach well software must ensure that beach well actuators, such as the valves and the pump, are controlled in a coordinated fashion and must function together to optimize the desalination process and avoid damage to the plant (e.g., when the beach well pumps run dry). To this end, the beach well software observes and orchestrates the beach well components so that process requirements are satisfied in accordance with the physical process itself. Table 1 shows an excerpt of such process requirements for a beach well.

Table 1. Process requirements for a beach well

ID	Process Requirement
1	Every active beach well must deliver a water flow of 400 m³/h to 750 m³/h
2	The pump may only run if filling level of the beach well tank is sufficient
3	The pump load shall be minimized using the bypass control valve
4	The discharge valve must be closed before pump starts
5	The discharge valve must be open after pump has started
6	The discharge valve must be closed after pump has stopped

To fulfill these process requirements, there is a number of signals exchanged between the beach well components and the beach well software. The signals are shown in Table 2. Signals

from the beach well and its components (Comp → BWS in Table 2) deliver information about the state of the beach well. They are used by the beach well software to control and coordinate the beach well (BWS → Comp in Table 2), depending on requests from the user (GUI → BWS in Table 2).

Table 2. Signals exchanged in order to control the beach wells

Component	Signal	Direction	Description
Beach Well	BWState	Comp → BWS	Current state of BW operation
	BWLevel	Comp → BWS	Water level inside the BW reservoir
	BPPressure	Comp → BWS	Water pressure in bypass pipes
	DCPressure	Comp → BWS	Water pressure in discharge pipes
Pump	PumpOnOff	BWS → Comp	Turns pump on or off
	PumpState	Comp → BWS	Current state of pump operation
	PumpSetPoint	BWS → Comp	Adjusts pump revolutions per minute
	PumpRev	Comp → BWS	Current revolutions per minute
Bypass Valve	BPVCommand	BWS → Comp	Requests to open or close the BPV
	BPVOpen	Comp → BWS	True, if bypass valve is fully open
	BPVClosed	Comp → BWS	True, if bypass valve is fully closed
Discharge Valve	DVCommand	BWS → Comp	Requests to open or close the DV
	DVOpen	Comp → BWS	True, if discharge valve is fully open
	DVClosed	Comp → BWS	True, if discharge valve is fully closed
User Interface	BWControl	GUI → BWS	User request to turn BW on or off
	Demand	GUI → BWS	User requested desired water
	Operation Information	BWS → GUI	Information about BW's current state

3 Solution approach

We seek to foster the early validation of automation control software by means of simulation during early stages of the plant development process, i.e., when the technical plant is not yet finished, but when assumptions about the technical plant architecture can be made. Figure 2 gives an overview of the proposed approach.

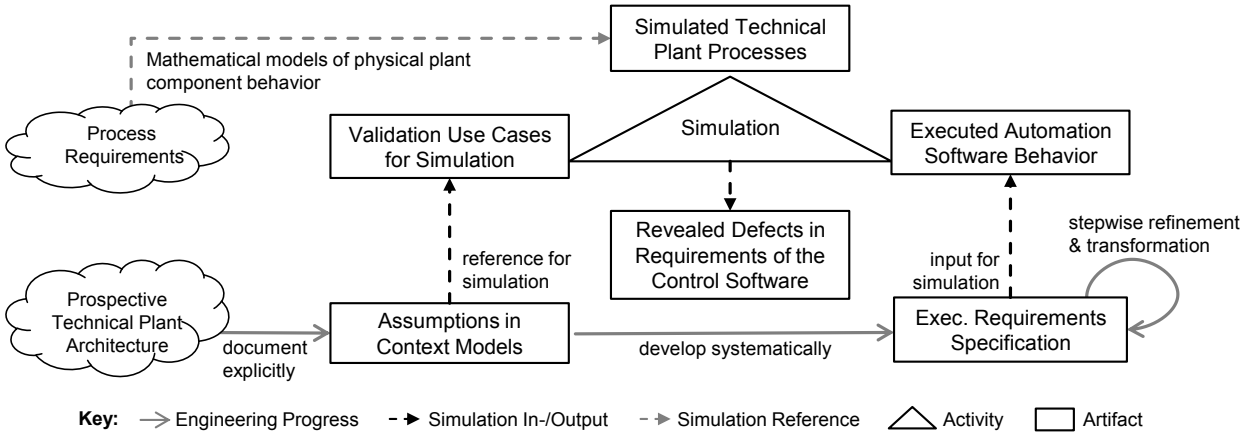

Figure 2. Overview over the approach

The key idea of our approach is to explicitly document the assumptions about the technical plant architecture by means of context models (see Section 3.1). Based on these context models, validation use cases can be developed (see Section 3.2) and an executable requirements specifi-

cation (see Section 3.3) can be systematically derived. Once a simulation tool has been configured based on the technical processes that are conducted in the plant (see Section 3.3.1), the behavior of the automation software can be executed (see Section 3.3.2) and checked against the validation use cases. The output of the simulation (see Section 3.3.3) is a set of revealed incorrect and incomplete behavioral requirements of the automation control software.

3.1 Assumptions about the prospective technical plant

The aim of our approach is to support developers in validating the proper functioning of the automation software with respect to the plant control requirements, which arise from the technical processes. In particular, our aim is to conduct validation early in the development process, i.e., when assumptions about the *intended* behavior of the *prospective* technical plant architecture can be made. In requirements engineering, such system properties are often documented in three distinct perspectives focusing on static-structural, functional, or behavioral properties of the system [4]. We adopt these perspectives to document assumptions about the technical plant architecture. Specifically, we employ a number of diagrammatic representations, which we call operational context models. The term operational context is motivated by the notion that the assumptions about the technical plant document properties of the assumed context of the automation control software (i.e., the physical plant components). The context models are the instrument to document and refine our understanding about the technical plant architecture, even if it is not completely defined or constructed. Figure 3 shows an excerpt of the overall ontological relationships between the different context model perspectives. In the following, we discuss the individual perspectives.

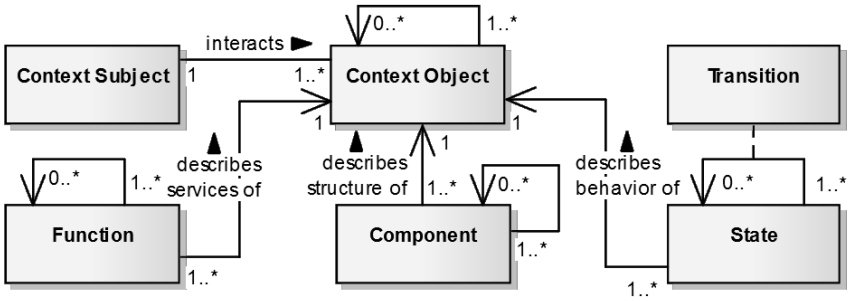

Figure 3. Ontology of the operational context model

Structural operational context. The structural operational context model documents structural characteristics of the beach well, its physical components, and the information exchanged between them and the beach well software [5]. This includes interfaces and interactions between human users and automation software of other plant components.

As shown in Figure 4, a beach well consists of a pump, a bypass valve, and a discharge valve. The signals introduced in Table 2 are represented through interfaces between the beach components and the beach well software. Since the structure and exchanged signals are the same for all four beach wells (see Section 2), the diagram only shows one beach well. In addition to the structure and signals, context influences are shown. For example, the diagram depicts that the beach wells pump water into a seawater tank and that the user receives information about the plant from the beach well software.

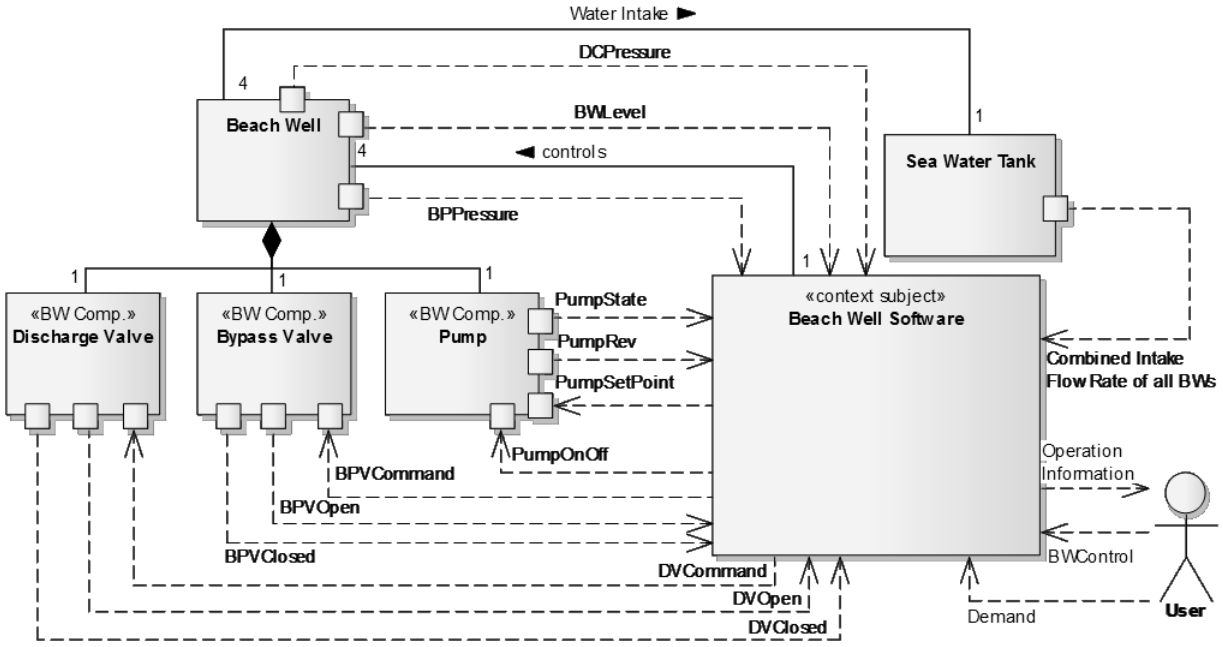

Figure 4. Structural operational context model of the beach well software

Functional operational context. The functional operational context model documents the externally visible functions of the automation control software and the physical plant components. In this model the focus is on the functional interplay between the automation control software and the physical plant components. This allows adopting a service-oriented view on the functionality by depicting only those functions that influence one another and is done by assigning concrete values to the signals from the structural operational context model that drive this influence. Figure 5 depicts the functional operational context model of the beach well software functions and the beach well component functions.

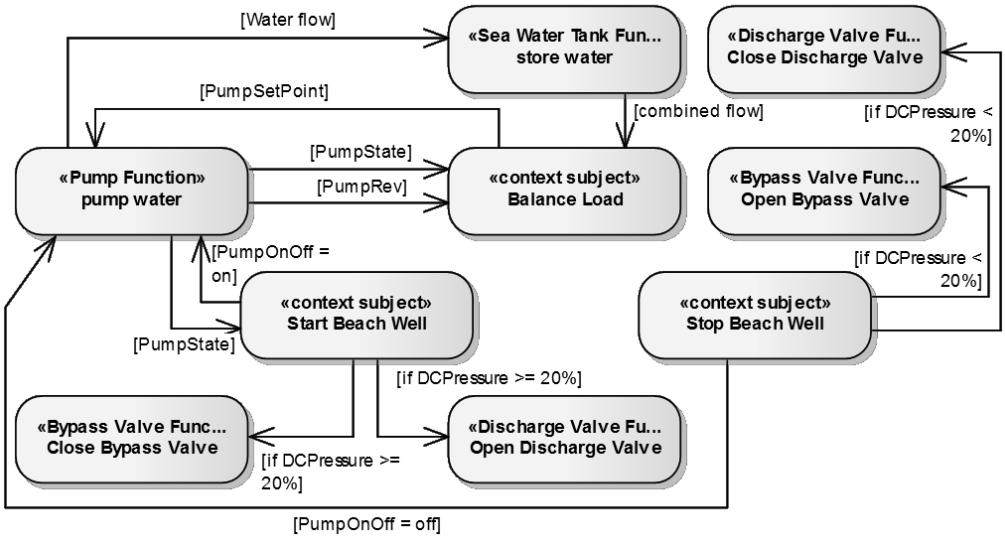

Figure 5. Functional operational context model of the beach well software

It is to note that in contrast to traditional activity diagrams, where the focus lies on control flow and/or data, in the functional operational context, the functional dependencies are of interest [6]. Therefore, albeit the notational elements of activity diagrams can be used to depict the functional operational context, entry and final pseudo states may be omitted. Furthermore, the guards on the activity edges in Figure 5 do not represent decisions in the control flow centric sense, but

conditions that the signals from the structural operational context must satisfy in order to meet the process requirements when controlling the plant.

As can be seen in Figure 5, the beach well software offers three externally visible functions, through which the beach well component functions are controlled: “start beach well”, “stop beach well”, and “balance load” (marked as *<<context subject>>* functions). The signals from the structural operational context model of the beach well software (see Figure 4 and Table 2) have been assigned with concrete values: The function “start beach well” shall only close the bypass valve and open the discharge valve if the pump has built up sufficient water pressure, such that water can flow into the seawater tank. Similarly, the function “stop beach well” turns the pump off, opens the bypass valve, and closes the discharge valve when water pressure is less than 20% of the maximum pressure.

Behavioral operational context. The behavioral operational context model documents the externally observable states of the physical plant components. In this sense, internal states of the plant components are abstracted and only the states relevant to the automation control software are depicted. Transitions between these states are triggered by values of the signals from the functional operational context.

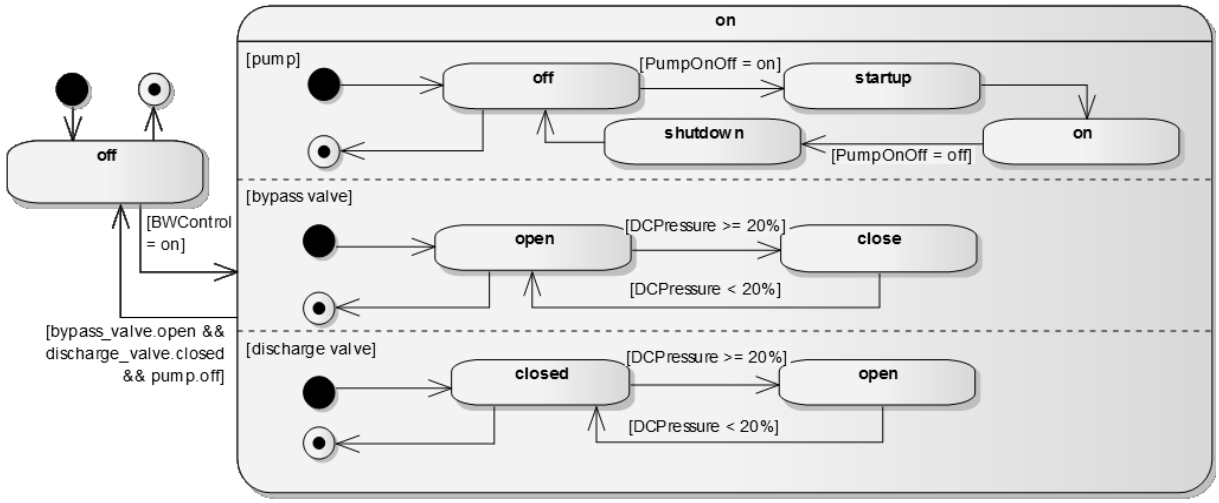

Figure 6. Behavioral operational context model of the beach well software

Figure 6 depicts the externally observable states of the beach well components from the structural operational context (i.e., the bypass valve, the discharge valve, and the pump) as concurrent substates of the entire beach well. The guards on the transitions are conditions specified in the functional operational context model with respect to the beach well function “start beach well”. Both bypass valve and discharge valve can be either open or closed, depending on the water pressure level (see Figure 5). The pump can be off, on, or in an intermediate state (startup or shutdown). The transition between “pump.startup” and “pump.on” does not have a guard. This indicates that, although there is an event that triggers the transition, the condition for this event is private to the pump and not externally visible. In other words, the beach well software can only observe the states, but has no influence on when this transition occurs.

3.2 Validation use cases

Based on the documented assumptions about the technical plant architecture by context models (see Section 3.1), validation use cases can be developed for every automation control software function defined in the functional operational context model. Validation use cases reference the externally visible states of the physical plant components from the behavioral operational context models, which serve as pre- and post-conditions. These are used as acceptance criteria under which the behavior of the automation control software is considered valid with regard to the pro-

cess requirements. Typically, validation use cases comprise a number of scenarios, which are sequential steps of interaction between the automation control software and the physical plant components [7]. They contain one success scenario (i.e., the sequence of steps leading to the desired post-condition), a number of alternative scenarios (i.e., alternative interactions leading to the same post-condition), and a number of exception scenarios (i.e., undesirable interactions, which violate the post-condition). For our beach well example, we develop the validation use case "Start Beach Well", as shown in Table 3.

Table 3. Validation use case "Start Beach Well"

Title	Start Beach Well	
Description	A beach well is manually started by the user of the desalination plant	
Rationale	Starting a beach well must follow a specific protocol that needs to be maintained by the beach well software	
Trigger	*BWControl == "The user initiates the start of the beach well".*	
Pre-condition	*bypass_valve.open == true && discharge_valve.closed == true*	
Post-condition	*pump.on == true && discharge_valve.open == true && bypass_valve.closed == true*	
Success Scenario		
Step	**Action**	**Actor**
1	The User initiates the start of a beach well	User
2	The Beach Well Software checks whether the beach well is in standby	Beach Well Software
3	The Beach Well Software closes the discharge valve	Beach Well Software
4	The Discharge Valve sends feedback that the valve is closed	Discharge Valve
5	The Beach Well Software starts the Pump with minimal revolutions	Beach Well Software
6	The Pump sends feedback that the Pump is started	Pump
7	The Beach Well Software closes the Bypass valve	Beach Well Software
8	The Beach Well Software opens the Discharge Valve	Beach Well Software
9	The Discharge Valve sends feedback that the valve is opened	Discharge Valve
10	The Beach Well Software reports that the beach well is on	Beach Well Software

The purpose of this validation use case is to operationalize the requirements from Table 1, specifically requirements 2, 4, and 5. We document validation use cases by templates that contain the essential information about this use case in accordance with [7]. For brevity, only the main scenario is shown and the alternative and exception scenarios have been truncated. As can be seen, requirements 4 and 5 from Table 1 are pre- and post-conditions regarding the startup procedure for the beach wells. The concrete conditions have been extracted from the behavioral operational context model (printed in italic in Table 3). The trigger of this validation use case relates to a signal from the user interface to the beach well software, depicted in the structural operational context model. Since the process requirements 2, 4, and 5 imply a certain sequence of valve actuation and minimal tank filling level, the beach well software must perform several sequential steps, documented in the success scenario.

To execute the scenarios from the validation use cases during simulation of the beach well software, a more formal representation is required. To this end, every scenario of each validation use case is refined into a Message Sequence Chart (MSC) [8]. MSCs describe the informal steps from the scenarios by means of formal messages between the automation control software and the physical plant components. Figure 7 shows the formalization of the success scenario from Table 3.

In this formal representation, the interface information from the structural operational context model (see Figure 4) is transferred to a sequence of formal messages. These messages trigger transitions from one state to another. These states correspond to the externally observable states of the physical plant components from the behavioral operational context model (see Figure 6). This formalized scenario serves as a reference for the simulation of the beach well software (see Subection 3.3.3).

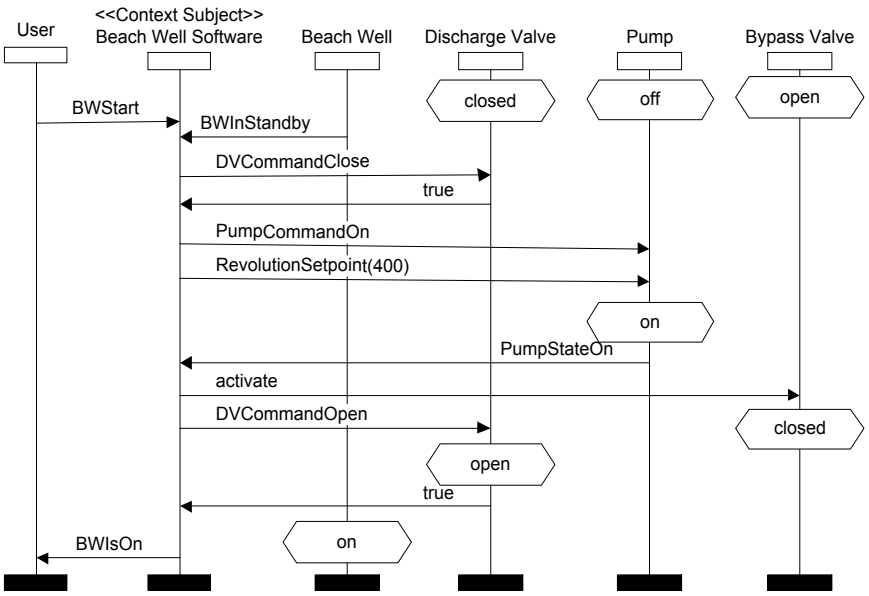

Figure 7. Success scenario from the validation use case "Start Beach Well" in Table 3

3.3 Executable requirements specifications

After the validation use cases are documented and their scenarios are formalized, a simulation tool can be configured (e.g., CoSMOS [9], Aspen, Simulink, or Modelica). The idea is that we try to reenact the validation use cases in a simulation of the automation control software and the technical plant architecture. For this purpose, the technical plant behavior and the automation software behavior must be modeled in an executable way (see Subections 3.3.1 and 3.3.2, respectively). Furthermore, both must be coupled in a simulative process, which executes the validation use cases (see Section 3.3.3).

3.3.1 Modeling the technical plant behavior

We model the technical plant behavior by describing the physical plant components by mathematical models, typically, differential/algebraic sets of equations (DAE). In the automation industry, these processes can often be composed from component libraries containing mathematical behavior models for generic physical plant components. The relevant configuration parameters (e.g., height of tank, length of pipes, etc.) of each component can be modified for different plant instances. Figure 8 shows how a technical plant architecture model with a beach well tank, a pump, a discharge valve, and a seawater tank is transformed into a simulation model.

The upper section of Figure 8 depicts the relevant excerpt of the technical plant architecture for our validation use case. Alongside the pump and the discharge valve from the structural operational context model (see Figure 4), which are controlled by the beach well software, the beach well intake tank and the seawater tank are depicted. All components can be found in the piping and installation diagram (P&ID, see Section 2). The middle section of Figure 8 shows the differential equations for flow, filling level, and pressure for both the beach well tank and the seawater tank. The behavior of the pump and the discharge valve is described by equations for flow and pressure. Since the four components are connected by three connections (which in a commissioned plant may correspond to water-bearing pipes), six additional equations are necessary to describe the flow balance in the physical pipes and the pressure potential at the connection points. In total, this system contains 14 equations and 14 variables.

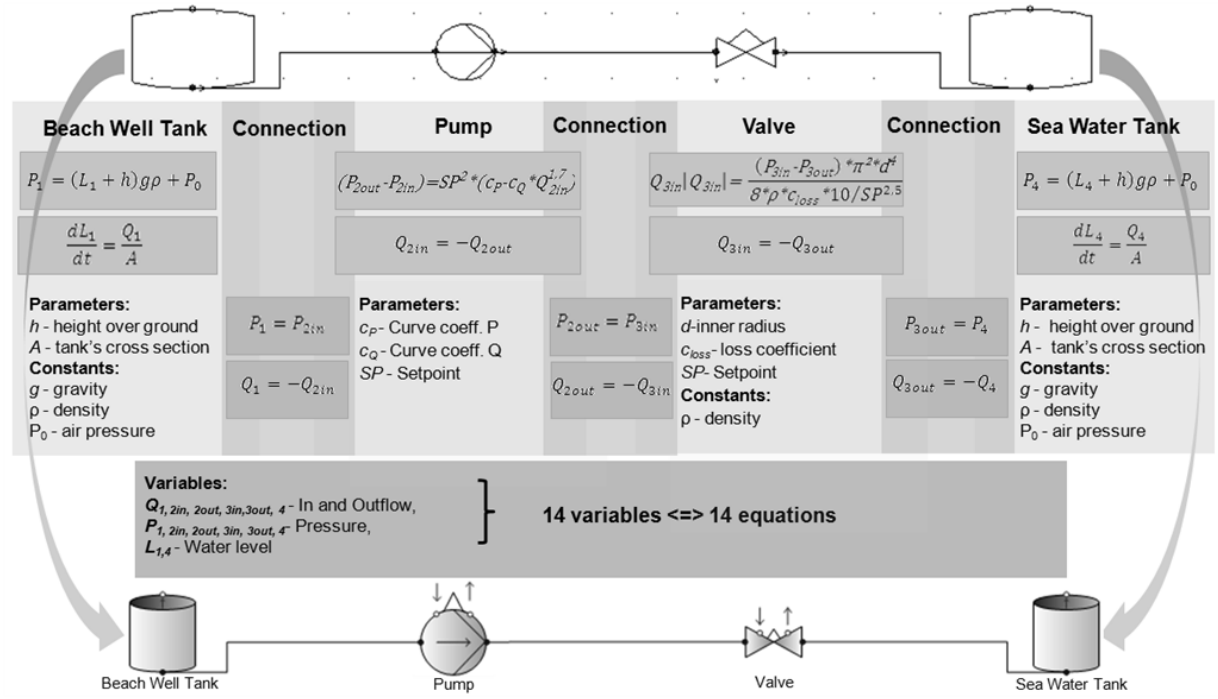

Figure 8. Generation of a technical plant behavior model from a technical architecture model

Once the relevant physical plant components are identified by means of the structural operational context model and once the equations representing their behavior have been compiled, the simulation tool can be configured, as indicated by the rounded arrows on the left and ride side of Figure 8. In our example, we use the tool CoSMOS [9], which uses icons similar to the P&ID (see the lower section of Figure 8). Using this tool, the plant behavior can be simulated by solving the resulting DAE-system.

3.3.2 Coupling with the executable automation software behavior

As a counterpart of the technical plant behavior, we model the functionality of the automation software to validate the behavior that emerges from the interaction between the automation software and the technical plant it controls. Therefore, we structure the functionality of the automation control software by means of functions [10]. A function summarizes a set of requirements concerning the user interaction and formalizes them by means of an executable behavior description. Figure 9 shows the two functions we defined for the beach well software.

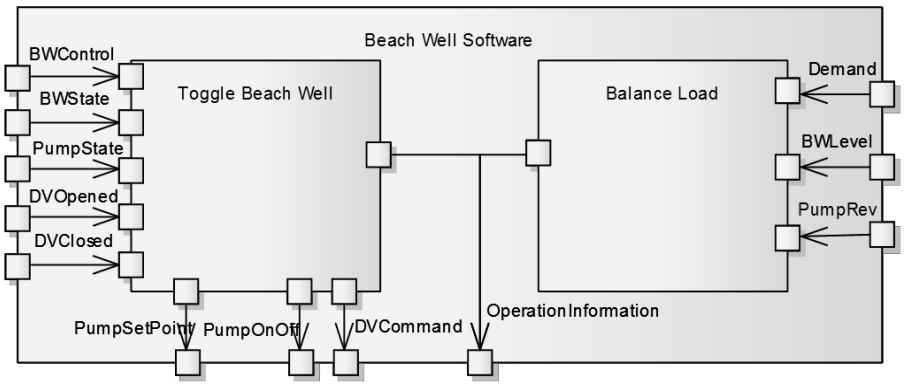

Figure 9. SysML block definition diagram showing two beach well software functions

For the specification of the behavior described by the validation use cases, we define two functions: “toggle beach well” and the “balance load”. Each of these two functions handles a subset of the input and output signals of the automation software that are specified in the structural operational context model (see Figure 4). In this case, both functions are independent from each other, but can be used in coordination. In an automation plant with a higher degree of automation, for example, if the function “balance load” may automatically start or stop additional beach wells by activating the function “toggle beach well”, it is necessary to add internal channels that model the communication between functions [11], [12].

Functions can be decomposed into further sub-functions or their behavior must be specified in terms of an executable behavior descriptions (e.g., state machines, a table specifications, or a code snippets). The reason for this is that we want to use the system specification in a simulation to execute the scenarios from the validation use cases. Figure 10 shows a state machine that describes the behavior of the “toggle beach well” function, which can be used to execute the success scenario from Figure 7.

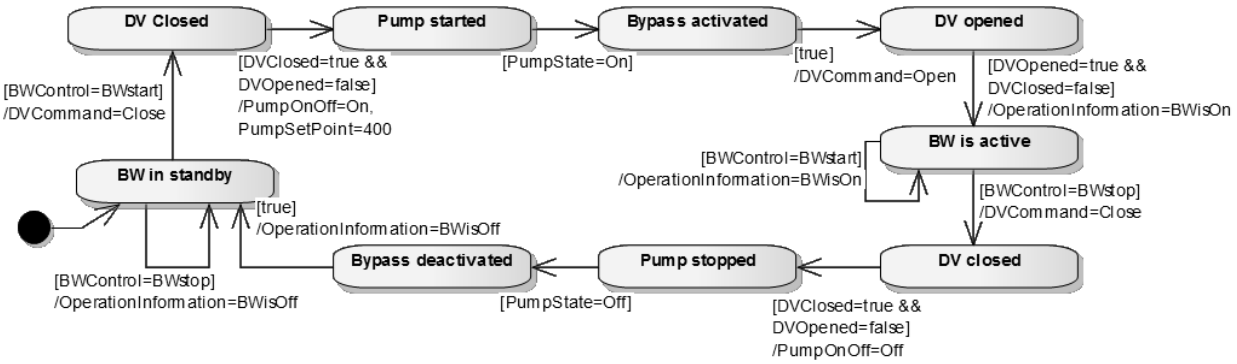

Figure 10. Behavior of the “Toggle Beach Well” function described by a state machine

3.3.3 Conducting the simulation

Based on the formalized validation use cases (see Section 3.2), the equation system documenting the technical plant behavior (see Section 3.3.1), and the automation software behavior (see Section 3.3.2), the simulation can be conducted. The simulation tool concurrently executes the behavior models of the automation software and the technical plant. The validation use cases and their associated scenarios are used as input for the simulation. If the simulation tool is able to execute the scenarios and the externally observable states from the behavioral operational context models match the final states of the physical plant components by the end of the simulation, the requirements specification is valid with regard to that use case. If the simulation tool is unable to execute the validation scenario and/or the post-conditions do not match the assumed externally observable states of the physical components, this is an indication that there is a defect. This defect may comprise incorrect or incomplete automation control software requirements, incorrect assumptions in the context models, or erroneously configured simulation parameters.

We implemented the approach using the academic tool AutoFocus3[1] to develop the context models, the validation use cases, and the executable automation software specification. We used the industrial tool CoSMOS to create an executable model of the technical plant process. To perform the validation process, we coupled the tools in the following way (see Figure 11). CoSMOS offers an open client-server co-simulation architecture. In this architecture, the co-simulation server orchestrates the interactions between simulation clients. Therefore, we implemented client instances for the AutoFocus3 simulation of the automation software, and for the CoSMOS simulation of the technical process. The goal of the simulation is to reenact specified validation sce-

[1] http://af3.fortiss.org/

narios. For this purpose, the co-simulation server performs one time step at every predefined sampling point in the automation software simulation, which is a discrete model. This means that the automation software is evaluated only on predefined time steps. The output then serves as an input for the continuous model of the technical plant simulation, which then covers the time between two steps of the discrete sampling points.

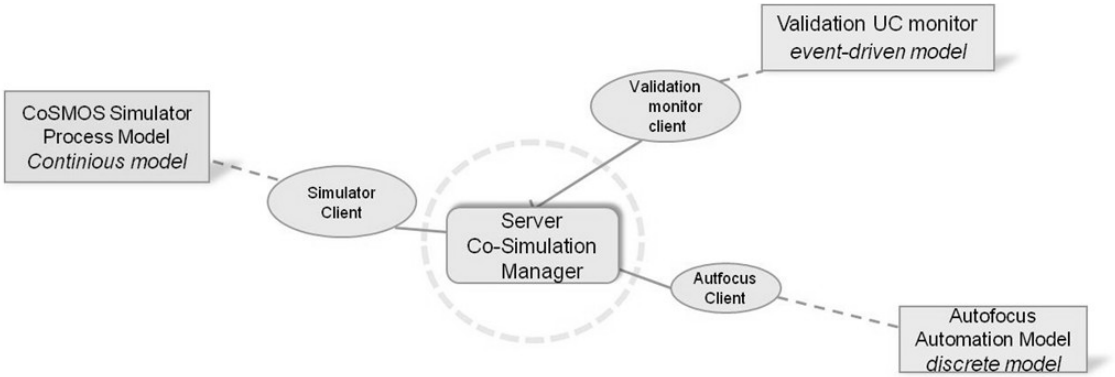

Figure 11. Implementation of the Co-Simulation Process

On top of this coupled simulation process, there is the third component also implemented as a simulation client called *validation use case monitor*. We feed this *validation use case monitor* component with the validation use case scenarios from Section 3.2. Technically, the validation use cases are translated to an XML file by expressing each scenario step as a simple *condition/action* rule (see the left side of Figure 12). This event-driven model monitors the simulation process and injects messages according to specified *condition/action* rules. The *validation use case monitor* is initialized with the first rule (i.e., scenario step) marked as "active". When the condition of this active rule is fulfilled by the running simulation, the corresponding action injects messages to the simulation and the following rule within the *validation use case monitor* is marked as "active". After the last rule (i.e., the last scenario step) has been processed, the *validation use case monitor* reports that the validation use case scenario has been reenacted by the simulation. If a currently active rule is not processed for a certain duration (timeout), then the validation use case monitor reports that the validation use case scenario could not be reenacted and therefore, the validation has failed. Figure 12 shows a visualization of the technical plant behavior over time that reflects the desired behavior as specified by the success scenario of validation use case "Start Beach Well".

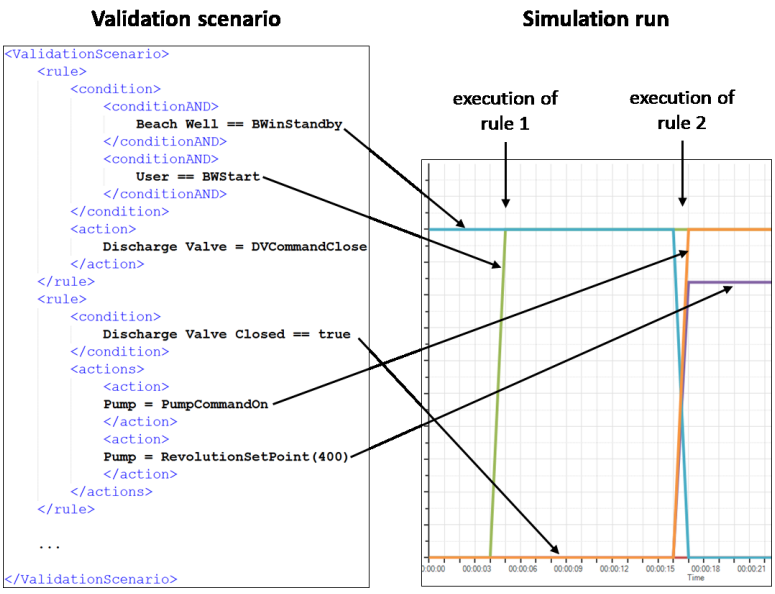

Figure 12. A validation scenario encoded in XML is reenacted in a simulation run.

Usually, in automation plant development projects, the technical plant behavior and layout are developed before the automation software is defined and the focus of this article is on conducting a coupled simulation of both. However, in situations where the technical plant behavior has not been defined yet, it is possible to use formal verification techniques (e.g., model checking) to check whether the discrete automation software model alone is able to fulfil the validation use cases [13].

4 Experiences from application in industry

As part of a broader research agenda, we applied the approach presented in Section 3 in detail to the desalination plant in a case study with partners from the automation industry. The case study has shown that the context model and its three perspectives provide useful information to capture the context information relevant to the software early in the engineering process. Context modeling leads to seamless and consistent models because information that was systematically derived and documented in context models can be referenced in the validation use cases. This leads to an improved tracing between development assumptions and requirements specification and makes validation criteria more objective. Relevant interfaces of the automation control software with the technical plant components are identified early. This is especially the case with respect to structural interfaces and functional dependencies between automation software and plant components, as identified in structural and functional operational context models. Additionally, the behavioral operational context model provides pre- and post-conditions for validation use cases.

The case study has furthermore shown that our approach allows the generation of executable models for the technical process and for the automation software. Before using our approach, the technical process and the executable models were described in inherently different structures, since the technical process describes a continuous system, whereas the automation control software is discrete following a predefined sampling rate. As both models have predefined interfaces, it is possible to couple them in a simulation model. This co-simulation process of animating the executable model fosters traceability between process action and software reaction, allowing developers to better judge the validity of the software. For example, when we conducted the simulation, the “start beach well” validation scenario failed initially, because closing the bypass valve and afterwards opening the discharge valve lead to overpressure in the pipes. As a result, we changed the automation software to close the bypass valve and open the discharge valve simultaneously, which lead to a successful simulation run.

Besides that, our approach allows the definition of basic constraints for the software design, such as control parameters, lag time, or limits. With respect to the running example, this might mean that a tank must not run dry and, therefore, it has to be ensured that a filling level of at least 10% must be maintained. In the coupled simulation process of our case study, we were able to reenact the specified validation use cases successfully.

Traditionally, placing particular emphasis on documenting context information that is shared among the different development disciplines involved in the automation industry is a novel concept in the development of automation control software. Interfaces, such as the signal list from Table 2, are usually defined individually and agreed upon in cooperation between two disciplines. In recent years, there has been a trend towards lifecycle engineering tools, which document plant data in an object-oriented and cross-disciplinary fashion. The approach we describe here can be applied preliminary to these lifecycle engineering tools. In this way, interfaces between different disciplines can already be set up and code segments can be generated from (semi-)formal models (e.g., UML or PCS7).

5 Related work

The related work, regarding the approach presented in Section 3, concerns two main topics: assumption and context modeling and requirements validation by means of simulation. These are discussed in the remainder of this section.

In requirements engineering, explicitly documenting implicit knowledge about the problem domain is seen as a prerequisite for structured validation. To this end,, in [10] and [14], the documentation of state-based behavior of the operational context is proposed, which allows documenting the operational context in the executable form. Similarly, in [16], representation techniques to document entities in the context of a system are suggested. Other approaches (e.g., [17], [18], [19]) have been proposed, which use documented context information for quality assurance, e.g., by evaluating development artifacts against assumptions about the context that may hold during runtime (i.e., goals in [17] and the world in [18]), or analyze the impact of context changes (see [19]). In all approaches, documenting context information is used as a reference artifact, against which other development artifacts can be validated. Therefore, we have incorporated context modeling as a central part of our approach such that assumptions about the business process (information systems) or the physical production process (in the automation industry) can be captured so that validation can take place as early as possible.

Most approaches regarding requirements validation by means of simulation are typically centered on animating requirements. In many of these approaches, scenarios are used to structure the interaction between users and the system under development (e.g., [20], [21]). However, in the case of information or automation control software, it must be validated that the behavior of the system satisfies process requirements, regardless if these requirements are imposed by technical processes or business processes. To this end, there are a number of approaches that deal with constraint satisfaction during dynamic execution of requirements. For example, in [22], formal requirements are executed and checked against a mock-up of the operational environment of the system. Similarly, in [16], [23], [24], and [25], behavioral models are used to derive an executable specification of the system under development. The key idea of these approaches is to describe the system in terms of a language with execution semantics, which is necessary for the simulation of automation control software in particular. To this end, Message Sequence Charts (MSCs) can bridge the gap between intuitive scenario-based visual notations and formal semantics. In our approach, we have therefore chosen MSCs to derive executable requirements systematically, which we subject to simulation.

6 Conclusion

In this article we presented an approach that foster the validation of automation control software against the technical plant process and the automation plant design at an early stage of development. Nowadays, the control software of automation plants is typically validated late in the engineering process, which may lead to serious budget overruns and projects delays in case the defects are revealed. Our approach enables the engineers to validate the application control software early in the engineering process as it validates the requirements of the automation control software against the technical plant process and assumptions about the design of the automation plant. For that purpose, the requirements of the control software are executed against assumptions and the technical plant process within a simulation environment in order to reveal defects in the executable requirements. It must be emphasized that, in our approach, the subjects of validation are the executable requirements derived from the context model of the automation control software and not the automation control software itself. Consequently, if no defects are revealed in the simulative validation, it can only be concluded that the requirements of the control software are valid with respect to the technical plant process and the assumptions about the plant

design. Under these circumstances, we can conclude that the corresponding automation control software will be valid if requirements have been implemented correctly.

Albeit this approach was developed in the context of the automation industry, we believe that it is applicable to any type of software system that controls real-world processes, such as real-time controlled feedback loops in case of embedded systems, or business processes in case of information systems.

Acknowledgements

This work was partially funded by the German Federal Ministry of Education and Research (BMBF), under grants number 01IS12005A, 01IS12005C, and 01IS12005R.

References

[1] U. Löwen, R. Bertsch, and B. Böhm, "Systematization of the engineering in automation plants," *Automatisierungstechnische Praxis*, vol. 4, 2005.

[2] T. Wagner, J. Wehrstedt, U. Löwen, T. Jäger, A. Fay, and P. Schuller, "Application and evaluation in the automation domain," in *Model-based Engineering of Embedded Systems*, Berlin Heidelberg, Springer, 2012. Available: http://dx.doi.org/10.1007/978-3-642-34614-9_11

[3] V. Brandstetter, A. Froese, B. Tenbergen, A. Vogelsang, J. Wehrstedt, T. Weyer, "Early validation of control software for automation plants on the example of a seawater desalination plant," in *Proc. the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015 Forum)*, 2015.

[4] A. M. Davis, *Software requirements: Objects, functions, and states,* 4th ed. Englewood Cliffs, NJ, PTR Prentice Hall, 1993.

[5] M. Daun, B. Tenbergen, T. Weyer, "Requirements viewpoint," in *Model-based Engineering of Embedded Systems*, Springer, pp. 55-58, 2012. Available: http://dx.doi.org/10.1016/j.scico.2010.06.00710.1007/978-3-642-34614-9_4

[6] M. Daun, A. Salmon, T. Weyer, "Using dedicated review diagrams to detect defective functional interplay in function-centered engineering," in *Proc. 5th Workshop on the Future of the Development of Software Intensive Embedded Systems (ENVISION 2020)*, pp. 31-40, 2015.

[7] A. Cockburn, *Writing effective use cases,* 16th ed. Boston, Addison-Wesley, 2006.

[8] ITU-T, *Z.120: Formal description techniques – Message Sequence Chart*, 2011.

[9] T. Schenk, A. Gilg, M. Mühlbauer, R. Rosen, and J. Wehrstedt, *Architecture for modeling and simulation of technical systems along their lifecycle*, Technical Report, TUM-NUM54, Technische Universität München, 2015.

[10] M. Broy, "Multifunctional software systems: Structured modeling and specification of functional requirements," *Science of Computer Programing*, vol. 75, no. 12, 2010. Available: http://dx.doi.org/10.1016/j.scico.2010.06.007

[11] A. Vogelsang and S. Fuhrmann, "Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study," *Proc. 21st IEEE International Requirements Engineering Conference (RE'13)*, 2013. Available: http://dx.doi.org /10.1109/RE.2013.6636728

[12] A. Vogelsang, H. Femmer, and C. Winkler, "Systematic Elicitation of mode models for multifunctional systems," in *Proc. 23rd IEEE International Requirements Engineering Conference (RE'15)*, 2015.

[13] A. Vogelsang, S. Eder, G. Hackenberg, M. Junker, and S. Teufl, "Supporting concurrent development of requirements and architecture – a model-based approach," in *Proc. the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD'14)*, 2014.

[14] T. Strang, C. Linnhoff-Popien, and K. Frank, "CoOL: A context ontology language to enable contextual interoperability," in *Lecture notes in computer science*, vol. 2893, *Distributed Applications and Interoperable Systems: 4th IFIP WG6.1 International Conference, 2003. Proc.*, J.-B. Stefani, I. Demeure, and D. Hagimont, Eds, Berlin Heidelberg, Springer, 2003. Available: http://dx.doi.org/10.1007/978-3-540-40010-3_21

[15] J. van den Bergh and K. Coninx, “CUP 2.0: High-level modeling of context-sensitive interactive applications,” in *Lecture notes in computer science*, vol. 4199, *Model driven engineering languages and systems: 9th international conference, MoDELS 2006. in Proc.*, O. Nierstrasz, Ed., Berlin, Springer, 2006. Available: http://dx.doi.org/10.1007/11880240_11

[16] K. C. Kang, G. J. Kim, J. Y. Lee, and H. J. Kim, “Prototype = function + behavior + form,” *SIGSOFT Softw. Eng. Notes*, vol. 23, no. 4, 1998. Available: http://dx.doi.org/10.1145/286366.286375

[17] A. van Lamsweerde, *Requirements engineering: From system goals to UML models to software specifications*. Chichester, Wiley, 2009.

[18] J M. Jackson, *Problem frames: Analysing and structuring software development problems*. Harlow, Addison-Wesley, 2001.

[19] L. de Alfaro and T. A. Henzinger, “Interface automata,” *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, 2001. Available: http://dx.doi.org/10.1145/503209.503226

[20] R. D. Acosta, C. L. Burns, W. E. Rzepka, and J. L. Sidoran, “A case study of applying rapid prototyping techniques in the Requirements Engineering Environment,” in *Proc. of the First Intl. Conf. on Requirements Engineering*, 1994. Available: http://dx.doi.org/10.1109/ICRE.1994.29240

[21] W. Dzida and R. Freitag, “Making use of scenarios for validating analysis and design,” *IEEE Trans. Software Eng,* vol. 24, no. 12, 1998. Available: http://dx.doi.org/10.1109/32.738346

[22] M. M. West and B. M. Eaglestone, “Software development: two approaches to animation of Z specifications using prolog,” *Softw. Eng. J. UK*, vol. 7, no. 4, 1992.

[23] A. M. Davis, “Rapid prototyping using executable requirements specifications,” *SIGSOFT Softw. Eng. Notes*, vol. 7, no. 5, 1982. Available: http://dx.doi.org/10.1145/1006259.1006266

[24] J. M. Thompson, M. P. E. Heimdahl, and S. P. Miller, “Specification-based prototyping for embedded systems,” in *Proc. of the Joint Conf. ESEC/FSE*, 1999. Available: http://dx.doi.org/10.1007/3-540-48166-4_11

[25] D. Li, X. Li, J. Liu, and Z. Liu, “Validation of requirement models by automatic prototyping,” *Innovations Syst Softw Eng*, vol. 4, no. 3, Available: http://dx.doi.org/10.1007/s11334-008-0062-3