Collaborative Model-based Development of a Remote Train Monitoring System

Peter Herrmann[1], Alexander Svae[1], Henrik Heggelund Svendsen[1] and Jan Olaf Blech[2]
[1]*Norwegian University of Science and Technology, Trondheim, Norway*
[2]*RMIT University, Melbourne, Australia*

Keywords: Cyber-physical systems, Software Engineering, Collaborative Development.

Abstract: The model-based engineering technique Reactive Blocks supports the development of reactive systems by UML-based graphic modeling of control and data flows, model checker supported analysis, and automated code generation. Moreover, it facilitates the cooperation of teams of engineers by enabling the definition of formally precise behavioral interfaces that make the separation of the modelling process into various work packages easy. In this paper, we illustrate the use of Reactive Blocks for a joint student project that realized the monitoring and control of Lego Mindstorms-based trains in Norway through a control center in Australia. In particular, we explain how the Reactive Blocks interfaces and the applied communication protocols were used to split the project into work packages separately handled by the students involved.

1 INTRODUCTION

Development, operation and maintenance of larger software systems is often done using teams of software engineers often working in distant places. Different companies may develop, deploy, maintain or operate different components of a system. Capabilities may be distributed over several locations, such as off-shore service centres and operations in other places. This is, e.g., the case in oil, gas and mining. In this paper, we regard transport systems which are spatially distributed. Trains and track controllers are inherently geographically separated from each other.

Well-defined responsibilities as well as clear interfaces between tasks help the work organisation and the documentation of system components for maintenance. The benefit may be increased by using modelling and formal techniques (Lee, 2008). For instance, with respect to software development in the transportation domain, we can distinguish two levels in which formal specifications can assist the collaboration of various stakeholders. On the *software component specification level*, formal specifications assist the engineering teams in developing, deploying and maintaining different software components. Often the teams need only limited coordination and can therefore be geographically distributed. In contrast, on the *operation model level*, the operation of trains may be formally modeled which assists us in conducting the control of a railway system in practice. Both levels have some interdependencies. For example, protocols may be derived from software component interfaces that influence the operation model level.

In this paper, we concentrate on the software component specification level and investigate the use of the model-based technique Reactive Blocks for cooperation of engineering teams. We illustrate the approach with the development of a distributed system that enables the remote monitoring and control of a Lego Mindstorms-based train system (Hordvik et al., 2016) residing in Trondheim through the visualization infrastructure VxLab (Blech et al., 2014) in Melbourne.

Below, we will discuss related work followed by an introduction to Reactive Blocks in Sec. 3 and a discussion in Sec. 4 how this technique can be used to coordinate different teams of engineers. Thereafter, we describe our train demonstrator in Sec. 5 and explain the ways, the involved students coordinated in Sec. 6. We complete the paper by a short discussion about future work.

2 RELATED WORK

Tool support for the collaboration of different stakeholders has been studied for various engineering do-

Herrmann, P., Svae, A., Svendsen, H. and Blech, J.
Collaborative Model-based Development of a Remote Train Monitoring System.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 383-390
ISBN: 978-989-758-189-2

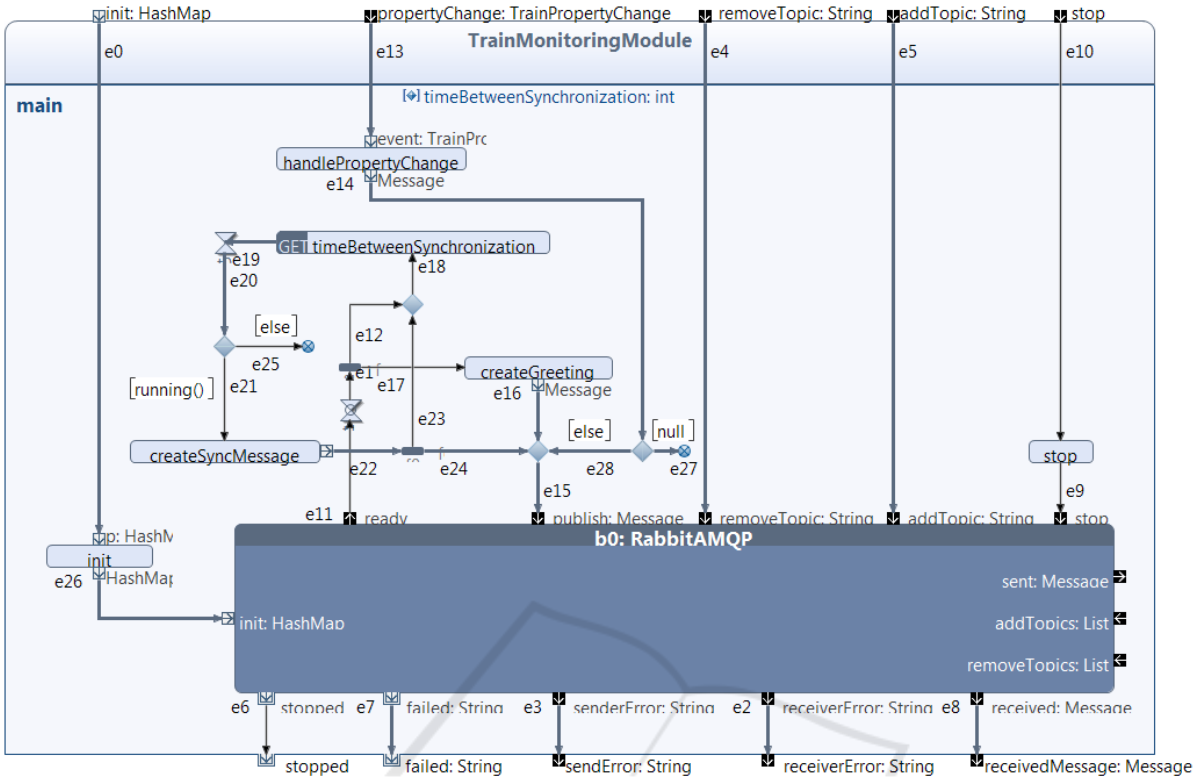

Figure 1: Building block *TrainMonitoringModule*.

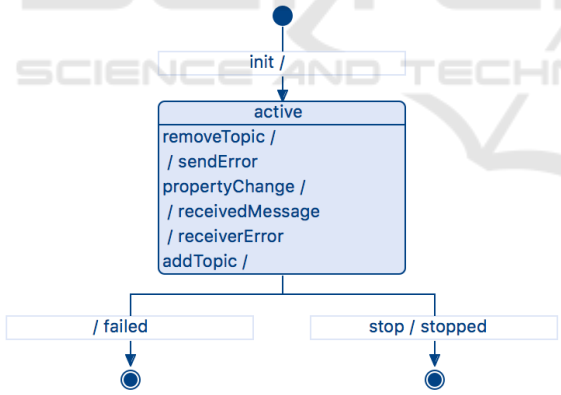

Figure 2: ESM of building block *TrainMonitoringModule*.

mains. Early examples comprise tool support for document sharing, e.g., (Toye et al., 1994) and work on collaborative software development environments, e.g., (Booch and Brown, 2003). Another study (Feiler et al., 2006) lists challenges for the development of ultra-large-scale systems. The analysis of collaboration patterns using social networks has been studied in (Cross et al., 2002). Results can be used to suggest team structures.

On the formal side of our work, methodologies for specifying components or work entities are of importance. Overall, we follow a design-by-contract approach (Meyer, 1992). We are primarily interested in contracts that specify the behavior of a component or an interface such as interface automata (De Alfaro and Henzinger, 2001). The work described in this paper integrates with behavioral types. Behavioral types have been used for the specification of real-time systems (Lee and Xiong, 2004). Furthermore, behavioral types are a formal description method that have been applied for software components (Blech et al., 2012), industrial automation systems (Blech et al., 2014) and cyber-physical systems (Blech and Herrmann, 2015). The idea is to have descriptions that allow for automatic checks of interactions between entities in very much the same way as type compatibility and compliance checking works in higher programming languages.

Furthermore, a connection of the work described in this paper to the collaborative engineering project is envisaged (Blech et al., 2015). Collaborative engineering provides means for different stakeholders to interact with each other focusing on maintenance and operation of large industrial systems.

3 REACTIVE BLOCKS

Reactive Blocks (Bitreactive AS, 2016; Kraemer et al., 2009) is a model-based engineering technique for reactive systems. It is Java-based and uses UML activities (Object Management Group, 2011) to specify control and data flow behavior. To make the approach scalable, an activity may be enclosed in a so-called *building block*. Further, one can represent a building block as nodes in other activities that are named *call behavior actions* in UML terminology. Thus, one can describe models as hierarchies of building blocks and their activities.

As an example, Fig. 1 depicts the activity of the building block *TrainMonitoringModule*. At its edges, the activity is provided by *parameter nodes* that are sources and sinks of flows, e.g., a parameter node `init` through which flows containing objects of the Java class `HashMap` may pass. The call behavior actions of a building block are endued with *pins* that correspond to the parameter nodes of its activity. For instance, our activity contains a call behavior action of building block *RabbitAMQP* that shows all the parameter nodes of its activity as pins (e.g., `init`, `ready`, and `publish`).

The parameter nodes of an activity are used to model flows from the activity of a building block to the one enclosing its call behavior action and vice versa. Our example specifies a flow that is started somewhere in the environment of building block *TrainMonitoringModule* and passes through pin/parameter node `init` of this block. Thereafter it continues in the activity in Fig. 1 and reaches the operation action `init` which is a container of a Java method of the same denominator being carried out when the flow passes. Afterwards, the flow reaches the pin of call behavior action *RabbitAMQP* such that it continues within the activity of the corresponding building block.

The building block concept makes it possible to specify functionality, that recurs in various applications, only once in one building block and to reuse its call behavior actions at various places. The Reactive Blocks tool contains hundreds of building blocks covering aspects reaching from flow control via communication protocols and encryption to domain-specific functions, e.g., for the Internet of Things, see (Bitreactive AS, 2016). To alleviate this reuse capability, each building block is accompanied by an External State Machine (ESM) (Kraemer and Herrmann, 2009). An ESM is a UML State Machine that models which pins/parameter nodes may be traversed in a certain state of the building block. As an example, we depict the ESM of building block *TrainMonitoringModule* in Fig. 2. It consists of two states, i.e., an initial one showing that the block is not active, as well as `active`. By tags on the edges, one describes which flows may appear in a certain state of the block and into which state the block changes. For instance, a flow through parameter node `init` is only allowed in the initial state and afterwards the block will be in state `active`. Tags within a state designator[1] (e.g., `removeTopic /`) describe that the corresponding flows may occur in the particular state but do not change the state. Several flows may enter and leave a block within a single atomic transition which is described by a list of parameter node identifiers (e.g., `stop / stopped`).

Reactive Blocks is provided by formal semantics (Kraemer and Herrmann, 2010) such that the tool uses a model checker verifying whether both, the activity enclosed in a block and the one using its call behavior action, indeed, comply with the ESM of the block. Moreover, the tool contains a code generator creating automatically executable Java code (Kraemer et al., 2006; Kraemer and Herrmann, 2007).

4 COORDINATING ENGINEERS WITH REACTIVE BLOCKS

The building block concept seems ideal to foster cooperative software engineering carried out by various teams of software engineers. In particular, it allows to structure the software architecture into several work packages. A building block can be used to define such a work package while its call behavior actions refer to places in which the results of this work package are used. The ESM of the building block provides then a behavioral interface description that facilitates the coordination between a team carrying out a work package and one using it.

The project planning corresponds to defining a hierarchy of building blocks that each describes a work package. In this phase, only an initial frame of a block is constructed. It does not contain the complete activity but only the parameter nodes to be used as well as the ESM. Moreover, the Java classes assigned to the parameter nodes are established at this stage.

After defining the building block hierarchy, the block frames are handed over to the various teams that develop the activities of the blocks and program the Java methods to which the operation actions of the

[1]The position of the / symbol describes whether a flow is triggered in the building block itself (left of the parameter node name, e.g., `/ failed`) or by its environment (on the right side, e.g., `init /`).

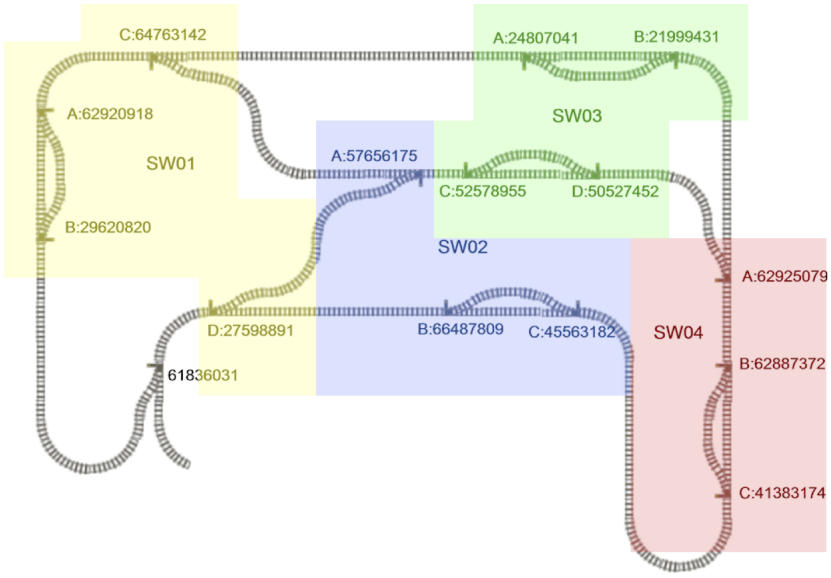

Figure 3: The Lego Mindstorms train layout used.

activity refers. Here, more complex behavior will lead to the separation of sub-functions in other building blocks that are also developed by the particular team or taken from the tool libraries. In addition, call behavior actions of blocks referring to other work packages may be used. When a team completes a building block, it verifies with the model checker of Reactive Blocks that their solution fulfills its ESM. Further, the model checker proves that the ESMs of the blocks referring to other work packages are preserved as well.

When all work packages are completed, the building blocks are assembled and the code for the entire system is generated. It can be used to test that the used objects conform with each other. Due to the comprehensible graphical UML models, these tests are usually easy.

The coordination of engineers is particularly important for geographically distributed systems. It profits from the fact that communication is traditionally service-oriented, see, e.g., (Tanenbaum and Wetherall, 2011). That means, distributed applications access the communication features via a *communication service*, that defines a set of functions like connection establishment or data transfer. The functions are provided by *communication protocols*. The engineers of different physical components can negotiate suitable communication services and protocols. They guarantee a correct communication by developing their applications such that the communication services are fulfilled.

Traditionally, communication services and protocols are specified formally using established techniques like SDL (ITU-T, 2011). Reactive Blocks is an alternative to these techniques. The local unit of a protocol can be realized as a building block while its ESM forms the corresponding communication service. For instance, the building block *RabbitAMQP*, used in the activity in Fig. 1, realizes a stack of the Advanced Message Queueing Protocol (AMQP), see (AMQP.org, 2016), that is popular in the Internet of Things domain. By following its ESM and by using the correct Java objects in the flows through its pins resp. parameter nodes, the correct usage of the protocol is guaranteed.

5 A TRAIN DEMONSTRATOR BASED ON REACTIVE BLOCKS

As mentioned in the introduction, a Lego Mindstorms-based train system, see (Hordvik et al., 2016), positioned in Trondheim, Norway, was connected to the visualization infrastructure VxLab (Blech et al., 2014) in Melbourne, Australia. Thus, the trains can be remotely monitored and, with some limitations, controlled. The work was done by some master student projects. One project (Svendsen, 2016) covered the development of the autonomous train control unit while a second one (Svae, 2016) realized the transfer of sensor data from Trondheim to Melbourne and the control commands the other way around. A third task, i.e., the access of the monitor and control data on the VxLab is still under progress.

In Fig. 3, we depict the layout of the tracks that connect five separate stations. The colors refer to

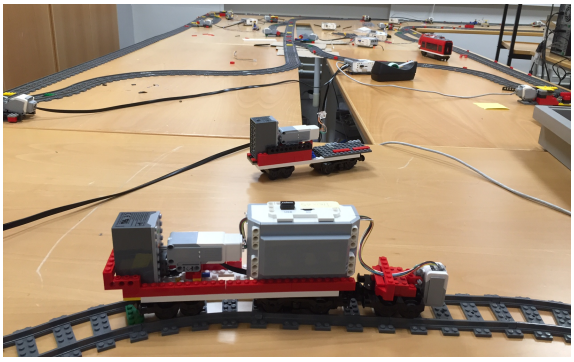

Figure 4: A Lego train.

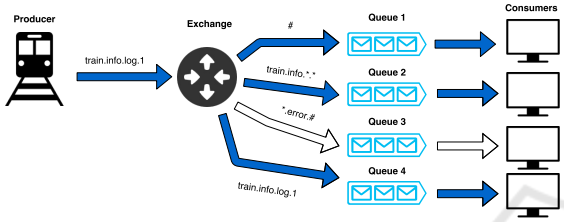

Figure 5: The remote AMQP infrastructure.

four Lego EV3 controllers that are used to operate the switches in the layout. A train includes a motor, an EV3 controller unit as well as a color sensor that may detect the color of the sleepers, the train passes (see Fig. 4). As described in (Svendsen, 2016), the trains operate autonomously. To guarantee a correct and safe operation, the EV3 controller of a train has to communicate with the controllers of the other trains as well as the ones operating the switches. For that, the AMQP protocol (AMQP.org, 2016) is used based on a local server.

To be flexible in defining track layouts, a layout to be used is modeled with the freeware layout editor BlueBrick (McKenna and Nanty, 2015). The model is automatically transformed into an internal map applied by the train controllers to state the current position of a train.

AMQP is also used for the remote monitoring and control of the trains (Svae, 2016). Besides to the local server, the EV3 controllers of the trains and switches keep also a connection to a remote AMQP server residing on the Australian cloud infrastructure Nectar (Nectar, 2016). Status information like the current position and speed of a train on a track or the settings of the switches is sent to the remote AMQP server allowing us to monitor train systems from the VxLab and other places “in the cloud”. Likewise, control commands from the VxLab are sent via the remote server to the train controller and can be directly executed. The used AMQP infrastructure is shown in Fig. 5.

```
{
    "id" : 1,
     "timestamp" : 1449832076300,
     "sequenceNumber" : 334,
     "event" : "SPEED",
     "speed" : 15
}
```

Figure 6: A JSON communication object example.

6 COLLABORATION BETWEEN THE STAKEHOLDERS

To coordinate the student projects, the interfaces between the work packages had to be defined. As discussed in Sec. 4, building blocks and communication protocols are suitable formal means to facilitate this coordination. We used both methods. For the transfer of monitoring and control data between Trondheim and Melbourne, the involved students agreed about the communication protocol and service to be used. To coordinate the control and remote communication parts within the EV3 controllers, the affected stakeholders negotiated a building block interface and the Java classes used in the control flows via this interface. Both interfaces are described in the following.

6.1 The Remote Control Connection

We discussed in Sec. 5 that the stakeholders agreed on using the Advanced Message Queueing Protocol (AMQP) (AMQP.org, 2016) with a remote server running in the Australian Nectar cloud (Nectar, 2016). Moreover, it was decided to use the JavaScript Object Notation (JSON) (ECMA International, 2013) as data interchange format. In JSON, information units can be defined as objects of sequences of pairs containing a name string and a value. Agreement on the exact JSON object formats for the various train parameters (e.g., train identifier, position, train length, speed, direction, switch position) as well as for the protocol control information (e.g., sequence numbers and time stamps) was reached.

Tests revealed that sending all train parameters in each communication message consumes much bandwidth. Therefore, the students decided to send only monitoring data that has changed in update messages. As an example, Fig. 6 lists the JSON object to be transmitted if a train changed its speed. Its pairs are the identifier of the train (`"id" : 1`), a time stamp and sequence number as protocol control information, event information (`"event : "SPEED"`), and the measured speed (`"speed" : 15`). To make the

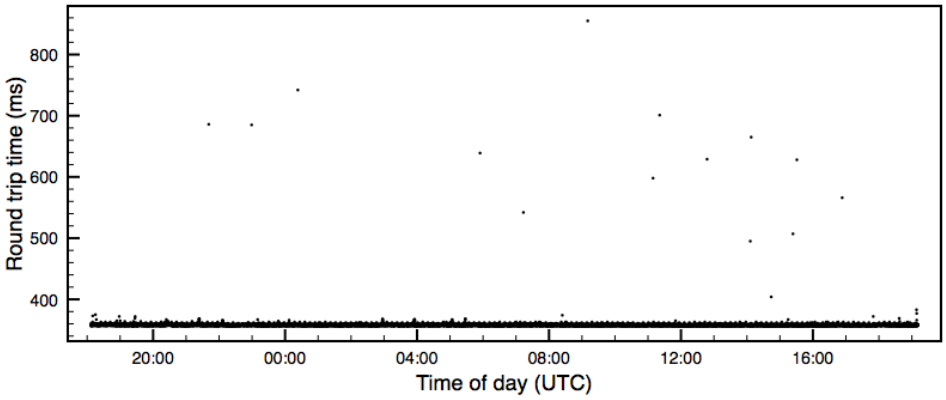

Figure 7: Measured round trip time delays between Trondheim and Melbourne.

communication more reliable, the update messages are accompanied by confirmed synchronization messages that are sent in certain time intervals. By checking the sequence numbers of the synchronization messages, one can find out which update messages were lost and resend the corresponding data. Further, remote control commands for the trains are sent via the AMQP link.

The handling of the update and synchronization messages in the EV3 controllers is realized by the building block *TrainMonitoringModule* (see Fig. 1). For instance, the Java method encapsulated in the operation `handlePropertyChangeAndCreateUpdateMessage` receives a list of all train parameters and checks which of them were changed since sending the last update message. For the altered parameters, an update message is created and sent via the communication block *RabbitAMQP*.

To find out about delays of the AMQP connection, the students carried out intensive round trip time tests. Figure 7 refers to a test checking if the round trip delay is fluctuating. For that, ping messages were sent every other second for 24 hours. As the figure shows, the delays were very stable between 350 and 360 ms with only very few fluctuations that never exceeded 880 ms.

6.2 Linking Train Control and Communication

The building block *TrainMonitoringModule* depicted in Fig. 1 defines the interface between the control and communication software part in the EV3 controllers. The involved students agreed on the data formats: The Java `hashmap` sent via parameter `init` contains the information necessary to build up AMQP connections. The Java class `TrainPropertyChange` used in parameter node `propertyChange` includes the relevant train and switch parameters to be send when updated. Messages received from the remote control are handed over to the block environment via parameter node `receiveMessage`.

Knowing about these formats as well as about the ESM of block *TrainMonitoringModule*, it was not difficult to build its call behavior action into the control software (Svendsen, 2016). The following issues had to be solved:

- Maintaining the link to the AMQP server in the Nectar cloud,
- after the change of at least one train resp. switch parameter, sending the parameter values in a Java object of class `TrainPropertyChange` via parameter node `propertyChange`,
- interpreting the JSON objects in messages, received via parameter node `receiveMessage`, and adjusting the train or switch control accordingly.

After incorporating the call behavior action of block *TrainMonitoringModule* and conducting the necessary changes, the model checker of Reactive Blocks verified that the ESM of this block is satisfied by the control software embedding it[2]. Thereafter, Reactive Blocks automatically generated an executable Java bundle that can be loaded into the EV3 controller and carried out there. Finally, some conformance tests were carried out revealing that the Java objects were correctly programmed.

7 CONCLUSION AND FUTURE WORK

We explained the capabilities of Reactive Blocks to facilitate collaboration for development and mainte-

[2]Further, the developer of block *TrainMonitoringModule* used the model checker to prove that the block fulfills its own ESM.

nance of software systems. In particular, we have been looking at transportation systems that, with respect to their software control, have a cyber-physical flavour. We demonstrated the capabilities using a remote train monitoring case study.

Future work comprises extensions for formalizing cyber-physical aspects and components and automatic tools to suggest tasks and supporting information for different distributed teams.

REFERENCES

AMQP.org (2016). Advanced Message Queuing Protocol (AMQP). www.amqp.org/. Accessed: 2016-02-01.

Bitreactive AS (2016). Reactive Blocks. www.bitreactive.com. Accessed: 2016-01-28.

Blech, J., Peake, I., Schmidt, H., Kande, M., Rahman, A., Ramaswamy, S., Sudarsan, S., and Narayanan, V. (2015). Efficient Incident Handling in Industrial Automation through Collaborative Engineering. In *IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*. IEEE Computer.

Blech, J. O., Falcone, Y., Rueß, H., and Schätz, B. (2012). Behavioral Specification Based Runtime Monitors for OSGi Services. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, LNCS 7609, pages 405–419. Springer-Verlag.

Blech, J. O. and Herrmann, P. (2015). Behavioral Types for Component-Based Development of Cyber-Physical Systems. In *Software Engineering and Formal Methods*, LNCS 9509, pages 43–52. Springer-Verlag.

Blech, J. O., Spichkova, M., Peake, I., and Schmidt, H. (2014). Cyber-Virtual Systems: Simulation, Validation & Visualization. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 1–8. IEEE.

Booch, G. and Brown, A. W. (2003). Collaborative Development Environments. *Advances in Computers*, 59:1–27.

Cross, R., Borgatti, S. P., and Parker, A. (2002). Making Invisible Work Visible: Using Social Network Analysis to Support Strategic Collaboration. *California Management Review*, 44(2):25–46.

De Alfaro, L. and Henzinger, T. A. (2001). Interface Automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120.

ECMA International (2013). Standard ECMA-404 — The JSON Data Interchange Format. www.ecma-international.org/publications/standards/Ecma-404.htm. Accessed: 2016-02-03.

Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Northrop, L., Schmidt, D., Sullivan, K., Wallnau, K., and Pollak, B. (2006). *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute.

Hordvik, S., Øseth, K., Blech, J. O., and Herrmann, P. (2016). A Methodology for Model-based Development and Safety Analysis of Transport Systems. In *11th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. To appear.

ITU-T (2011). Z.100 : Specification and Description Language - Overview of SDL-2010. www.itu.int/ rec/ T-REC-Z.100/en. Accessed: 2016-02-01.

Kraemer, F. A. and Herrmann, P. (2007). Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In *6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, Electronic Communications of the EASST 7. EASST.

Kraemer, F. A. and Herrmann, P. (2009). Automated Encapsulation of UML Activities for Incremental Development and Verification. In *Model Driven Engineering Languages and Systems (MoDELS)*, LNCS 5795, pages 571–585. Springer-Verlag.

Kraemer, F. A. and Herrmann, P. (2010). Reactive Semantics for Distributed UML Activities. In *Joint WG6.1 International Conference (FMOODS) and WG6.1 International Conference (FORTE)*, LNCS 6117, pages 17–31. Springer-Verlag.

Kraemer, F. A., Herrmann, P., and Bræk, R. (2006). Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In *8th International Symposium on Distributed Objects and Applications (DOA06)*, LNCS 4276, pages 1614–1632. Springer-Verlag.

Kraemer, F. A., Slåtten, V., and Herrmann, P. (2009). Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software*, 82(12):2068–2080.

Lee, E. (2008). Cyber Physical Systems: Design Challenges. In *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE Computer.

Lee, E. A. and Xiong, Y. (2004). A Behavioral Type System and its Application in Ptolemy II. *Formal Aspects of Computing*, 16(3):210–237.

McKenna, A. and Nanty, A. (2015). BlueBrick — Version 1.8.0. bluebrick.lswproject.com/help_en.html. Accessed: 2016-02-02.

Meyer, B. (1992). Applying "Design by Contract". *Computer*, 25(10):40–51.

Nectar (2016). NectarCloud. cloud.nectar.org.au/. Accessed: 2016-02-02.

Object Management Group (2011). OMG Unified Modeling LanguageTM (OMG UML), Superstructure — Version 2.4.1. www.omg.org/spec/UML/2.4.1/ Superstructure/PDF/. Accessed: 2016-01-28.

Svae, A. (2016). Remote Monitoring of Lego-Mindstorm Trains. Project thesis, Norwegian University of Science and Technology, Trondheim.

Svendsen, H. H. (2016). Model-based Engineering of a Distributed, Autonomous Control System for Interacting Trains, deployed on a Lego Mindstorms Platform.

Project thesis, Norwegian University of Science and Technology, Trondheim.

Tanenbaum, A. S. and Wetherall, D. J. (2011). *Computer Networks*. Prentice Hall, 5 edition.

Toye, G., Cutkosky, M. R., Leifer, L. J., Tenenbaum, J. M., and Glicksman, J. (1994). SHARE: A Methodology and Environment for Collaborative Product Development. *International Journal of Intelligent and Cooperative Information Systems*, 3(2):129–153.