Towards a Model-based Toolchain for Remote Configuration and Maintenance of Space-aware Systems

Jan Olaf Blech¹, Peter Herrmann², Ian Peake¹ and Heinz Schmidt¹

¹RMIT University, Melbourne, Australia
²Norwegian University of Science and Technology (NTNU), Trondheim, Norway

Keywords: Model-driven Software Engineering, Industrial Automation.

Abstract: We present work towards a toolchain that combines our existing tools Reactive Blocks and BeSpaceD with our remote collaboration and visualization facility VxLab. Software development in areas such as oil and gas, mining or automation is subject to remote configuration and maintenance of installations. Different reasons are driving this trend including difficult accessibility of remote sites and outsourcing to offsite experts or due to cheaper labor costs. Here, we concentrate on work towards remote configuration, installation and maintenance of the software controlling these installations and their spatial constraints.

1 INTRODUCTION

Industrial activities connected to natural resources often take place on remote locations, e.g., mining districts in the Australian Outback or oil rigs in the North Sea. To reduce the high expense for accommodation and transport of employees, the industry production on such places is increasingly automated and controlled, configured resp. maintained from a remote location.

In many cases, work is done by machines or robots that cooperate in close proximity to each other. Machinery has an influence on the physical space, for example by heat emission or occupying it physically, that can have an influence on the entire plant. The coordination of the machines as well as the necessity to achieve a good productivity lead to complex control software with extraordinary functional and safety requirements. Furthermore, due to the changing environment at, e.g., a mining area, the control software frequently needs to be configured in a customized way which leads to refactoring and adaptation. This calls for highly skilled software experts who could be permanently positioned at the remote spots at enormous costs only.

To cope with this challenge, we present initial work on the development of a toolchain allowing distributed development, adaptation, installation and maintenance of control software for space-aware cyber-physical systems. In particular, we aim at combining and amending three existing tools:

1. Reactive Blocks (Kraemer et al., 2009) is a model-based engineering technique for reactive systems.
2. BeSpaceD (Blech and Schmidt, 2014; Blech and Schmidt, 2013) is a tool for the verification of spatiotemporal properties.
3. VxLab (Blech et al., 2014b) is a technique for the cooperation of experts at various places in order to simulate, validate and visualize industrial processes.

As proof of concept, we create a toolchain that allows software developed by Reactive Blocks to control a pair of cooperating ABB IRB120 industrial robot arms that are located at RMIT University in Melbourne. To verify properties with respect to spatial behavior, we use the existing connection between Reactive Blocks and BeSpaceD (Han et al., 2014; Herrmann et al., 2014). Moreover, we extend the existing VxLab functionality in a way that the installation of the generated software in the control computers of the robots is directly supported. This allows, for instance, software experts located in Trond-
2 BACKGROUND INFRASTRUCTURE

Below, we sketch the model-based engineering method Reactive Blocks, the spatiotemporal verification tool BeSpaceD, and the virtual laboratory VxLab that form the ingredients of our toolchain.

2.1 Reactive Blocks

Reactive Blocks\(^1\) (Kraemer et al., 2009) facilitates the model-based development of reactive software. In particular, we have attached importance to reusing the models of certain sub-functionality that may appear in various applications of a certain domain. The partial models are realized as building blocks that each consists of a UML 2.x activity diagram modeling detailed implementation logic and an abstract External State Machine (ESM) (Kraemer and Herrmann, 2009) specifying the interface behavior of the building block. System models can be developed by taking building blocks from libraries, creating others oneself, and combining the various blocks using the operators of UML activities. Since we provided the UML activities and ESMs with a formal semantics (Kraemer and Herrmann, 2010), system models can be automatically analyzed for functional errors by a built-in model checker (Kraemer et al., 2009). An extended version allows also the verification of real-time properties (Han and Herrmann, 2013). The system models are automatically transformed into executable Java code.

2.2 BeSpaceD

In (Blech and Schmidt, 2014; Blech and Schmidt, 2013), we introduce BeSpaceD as a tool framework for specifying behavior of distributed cyber-physical systems and formally reasoning about them. BeSpaceD emphasizes on spatial behavior but is not restricted to this. It allows the verification of safety properties such as the absence of physical collisions between interacting robots and obstacles, the coverage of sensor ranges, or WLAN ranges. Specification is done using abstract datatypes out of a development environment supporting the Scala programming language. The abstract datatypes can be generated by Scala programs or by instantiation of other software. Operations such as checking and reasoning in BeSpaceD is realized using library functions creating verification goals. Verification goals are solved by standard tools such as SAT and SMT solvers or by specialized algorithms.

2.3 VxLab and Industrial Robot Control

Our Virtual “x” laboratory (VxLab) (Blech et al., 2014a) aims at enabling decision making and design

\(^1\)Before being marketed by BitReactive AS (http://www.bitreactive.com), the tool was named Arctis.
among leaders, experts and technicians distributed globally, over multiple use cases (signified by the “x” parameter) such as scientific computing, gaming, software development, engineering and architecture. VxLab is a generalization of x = “Interoperation Testing” realized in the VITELab, a predecessor eResearch facility of the Australia-India Research Centre for Automation Software Engineering (AICAUSE)2. AICAUSE is a partnership between RMIT University and the ABB Groups in Australia and India with support from the Victorian State Government. VITELab is designed as a global “lab scope” connecting industry and university sites to enable collaboration for experimental design and testing of distributed cyberphysical systems.

VxLab includes, among other facilities, the Global Operations Visualization (GOV) lab, providing a high resolution visualization wall with integrated video conferencing/streaming to remote sites, the Cyber-physical Simulation (CS) Rack, a blade server configurable via OpenStack (openstack.org), and a dedicated private network with high-speed connection to industry partners. Further, the Advanced Manufacturing Robot Interoperation Test (AMRIT) lab provides two ABB IRB120 robot arms with standard IRC5 controllers and Robotiq adaptive 3-fingered grippers, that will be used in our proof of concept. GOV lab uses SAGE visualization middleware3, to prototype next generation applications via “mashups” combining user interfaces of existing software (running on local, remote or even virtual hosts) with concept images/video.

A view of the AMRIT lab from the GOV Lab is shown in Fig. 2. One of the robots is depicted in Fig. 3. The GOV Lab has been applied to collaboratively develop, test and monitor cyber-physical applications remotely, such as in the AMRIT lab, with multiple users flexibly and simultaneously interacting with multiple applications/services, such as ABB’s RobotStudio IDE/Simulation tool and live views of robots. For example, we developed a concept demonstration where real and virtual robots interact in real time. A .Net application synchronizes, via the RobotStudio API, operation of two independent robot controllers, one real, one simulated. On the video wall, a simulated robot is positioned as an overlay over the live camera view in the place where its live counterpart would exist in a fully integrated system.

3  REMOTE DEVELOPMENT AND CONTROL OF ROBOTS

As discussed in Sect. 2.1, Reactive Blocks allows the creation of system models by composition of reusable building blocks. To exemplify the engineering process, we created a first Reactive Blocks model for the


remote software control part of the robot. A result is the UML 2.x activity diagram modeling the behavior of a building block GripCanSimple which is depicted in Fig. 4. This building block represents the task to grip a can, e.g., the white one shown in Fig. 3, with a robot arm. The activity uses four inner building blocks which are taken by simple drag-and-drop from a library.

The building block DirectTrajectory represents the movement of the grip from its present position and orientation on a linear trajectory to another one. UML activities model behavior similarly to Petri nets such that we can interpret behavior as a sequence of token flows. Block DirectTrajectory is started at the pin to and tokens passing this pin contain an object of class PositionData that describes the position and orientation to be moved to. When the robot arm has reached its new position and orientation, the block terminates via issuing a token through pin onPosition. Likewise, opening and closing of the grip can be realized using library blocks.

With these library blocks, it is relatively simple to create other building blocks modeling more complex robot behavior. As shown in Fig. 4, a proof-of-concept implementation for GripCanSimple is started by a token received through the parameter node, i.e., the pin at the activity edge, startTo that contains the position of the can to be gripped. The token forwards to a set variable action in which the position information is stored in the variable canPosition. Thereafter, it reaches operation toStartPoint that refers to a Java method of the same name which is triggered. By this method, the starting point of gripping the can is computed which is 30 cm above the can position with an orientation such that the grip points downwards. Afterwards, this position information is forwarded to an instance of building block DirectTrajectory that encapsulates the code to move the robot grip to the starting point. In the succeeding steps, the grip is opened and thereafter lowered to the position of the can. Finally, the grip is closed until a certain resistance is reached such that the can is solidly gripped without being deformed. When this step is finished, building block GripCanSimple terminates via sending a token to its environment through parameter node complete.

The building block GripCanSimple was checked for functional correctness by the built-in model checker of Reactive Blocks. In particular, it was verified whether the own ESM of the block as well as those of the inner blocks are satisfied (Kraemer et al., 2009). If required, one can further verify if the block fulfils certain real-time properties, e.g., that a can is gripped within a certain time interval (Han and Hermann, 2013).

Moreover, our building blocks can be proven with BeSpaceD for spatiotemporal properties (see Sect. 2.2). For example, one can verify whether block GripCanSimple guarantees that a can is gripped with-
out being previously overturned. This property does not hold since the grip can initially be very close to the can such that it can be touched on the trajectory to the starting point 30 cm above the can.

To avoid overturning, we replace building block GripCanSimple by GripCan that is depicted in Fig. 5. Here, we do not move the grip immediately to the starting point if it is close to the can but first increase the distance to the can. For that, we read the current position of the grip using the singleton block PositionHandler. Thereafter operation inVicinity checks if the grip is close to the can. If that is the case, the token leaves the decision node behind inVicinity through the edge with guard true to an instance of block DirectTrajectory that moves the grip to a position farer away from the can before it is moved to the starting point. For this variant, BeSpaceD can verify that the can is not overturned during gripping.

Similarly, we can create building blocks for the other functions of the robot and compose them to a Reactive Blocks model specifying the overall system behavior. When this model passes the correctness proofs, it can directly be transformed to Java code.

The main task for the proof of concept is to create the functionality of the building blocks in our library (like DirectTrajectory) such that the RobotStudio- and .Net-based RMIT robots can be directly accessed. For this, we want to use a robot configuration service that offers an API containing routines to instantiate robot movements. An initial version of this service is already tested and it is not difficult to call its routines from Java. Thus, after being started by flows via pin to, the building blocks DirectTrajectory, OpenGripper, and CloseGripper call routines of this service. Thereafter, they wait until receiving a confirmation message from the service which leads to terminating the blocks with flows through pins onPosition resp. complete. Further, the robot configuration service can be offered as a cloud based service such that it may also be operated from remote stations.

As in the existing .Net based solution, the Java code is only specifying the overall behavior and does not need to run on the robot system directly. It emits more detailed motion control commands which are stored and processed on a stack in the local robot controller. These local commands comprise exact rotation and movement information whereas the Java code is responsible for the control flow of the underlying application and communication with additional devices.

The system is tested and the results are visualized using VxLab. This includes camera feedback and access to sensor and actuator configuration information. Further, VxLab supports the management of the produced code and the robot configuration service.
4 CONCLUSION AND ONGOING WORK

We presented our ideas and first work towards a toolchain for developing robot control software. The toolchain comprises development using Reactive Blocks, spatial verification, remote deployment of control software as well as remote visualization and monitoring of the robots. As of now, Reactive Blocks, spatial verification using BeSpaceD and the remote visualization and monitoring via VxLab exist. Remote deployment and configuration of robots is ongoing work.

REFERENCES


dation & Visualization. In Evaluation of Novel Approaches to Software Engineering.


