Table of Contents

International Journal of Web Services Research

Volume 13 • Issue 1 • January-March-2016 • ISSN: 1545-7362 • eISSN: 1546-5004
An official publication of the Information Resources Management Association

Special Issue on New Techniques of Services Computing

Guest Editorial Preface

Research Articles

The *International Journal of Web Services Research* is indexed or listed in the following: ABI/Inform; ACM Digital Library; Bacon's Media Directory; Burrelle's Media Directory; Cabell's Directories; Compendex (Elsevier Engineering Index); CSA Illumina; Current Contents®/ Engineering, Computing, & Technology; DBLP; DEST Register of Refereed Journals; Gale Directory of Publications & Broadcast Media; GetCited; Google Scholar; INSPEC; Journal Citation Reports/Science Edition; JournalTOCs; Library & Information Science Abstracts (LISA); MediaFinder; Norwegian Social Science Data Services (NSD); PubList.com; Science Citation Index Expanded (SciSearch®); SCOPUS; The Index of Information Systems Journals; The Standard Periodical Directory; Thomson Reuters; Ulrich's Periodicals Directory; Web of Science

International Journal of Web Services Research
Volume 13 • Issue 1 • January-March 2016

A Model-Based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems

Peter Herrmann, Norwegian University of Science and Technology (NTNU), Trondheim, Norway
Jan Olaf Blech, RMIT University, Melbourne, Australia
Fenglin Han, Norwegian University of Science and Technology (NTNU), Trondheim, Norway
Heinz Schmidt, RMIT University, Melbourne, Australia

ABSTRACT

A method preserving cyber-physical systems to operate safely in a joint physical space is presented. It comprises the model-based development of the control software and simulators for the continuous physical environment as well as proving the models for spatial and real-time properties. The corresponding toolchain is based on the model-based engineering tool Reactive Blocks and the spatial model checker BeSpaceD. The real-time constraints to be kept by the controller are proven using the model checker UPPAAL.

KEYWORDS

BeSpaceD, Model-Based System Engineering, Reactive Blocks, Real-Time Properties, Spatial Behavior Modeling and Verification

1. INTRODUCTION

In safety critical domains like aviation, automotive and robotics, autonomous cyber-physical systems interact with each other in the same physical space. To avoid damage and injuries, the control software of the systems has to guarantee spatiotemporal properties like collision avoidance or the cooperation of several units that carry a heavy workpiece together. A popular way for the creation of functionally correct and safe system software is the application of integrated modeling and verification tools like MATLAB/Simulink (Tyagi, 2012). Our contribution is the combination of such a tool with efficient provers allowing to verify that the coordinated behavior of multiple controlled cyber-physical systems fulfills relevant spatial safety properties. We introduce a toolchain combining the model-based engineering tool-set Reactive Blocks[1] (Kraemer, Slåtten, & Herrmann, 2009) with the verification

DOI: 10.4018/IJWSR.2016010103

tool BeSpaceD (Blech & Schmidt, 2013). In particular, we use a development workflow starting with the collection of requirements for a cyber-physical system and its architecture followed by the steps listed below:

1. Spatiotemporal properties of components are described in the input language of BeSpaceD;
2. A model of the system controller is created in Reactive Blocks. We compose it with a simulator model of the continuous system parts which is created using the BeSpaceD model developed in step 1;
3. The built-in model checker of Reactive Blocks is used to check the combined controller and simulator model for general design errors (Kraemer, Slåtten, & Herrmann, 2009);
4. If the checks in step 3 are passed, the software model is transformed to the input language of BeSpaceD;
5. Assuming certain maximum reaction times of the discrete controller, it is verified with BeSpaceD that the model resulting from the transformation in step 3 fulfills the spatiotemporal properties defined in step 1;
6. The model checker UPPAAL (Bengtsson, et al., 1996) is applied to prove that the real-time properties assumed in the proofs of step 5 are indeed kept by the Reactive Blocks model created in step 2 (Han & Herrmann, 2013), (Han, Herrmann, & Le, 2013);
7. By using the code generator from Reactive Blocks (Kraemer & Herrmann, 2007), (Kraemer, Herrmann, & Bræk, 2006) executable Java code of the controller and, if desired, of the simulator of the continuous behavior is created. The generated code can be deployed on the system components running the control software of the embedded system.

Our approach has to guarantee that a model developed with Reactive Blocks indeed fulfills the desired safety properties if the verifications in steps 5 and 6 succeed. Formally, that proof is merely trivial: Be S the logical formula corresponding to a system model in Reactive Blocks according to (Kraemer & Herrmann, 2010), P the conjoined spatial behavioral properties to be fulfilled by S, and $R(t)$ a statement describing that the controller always guarantees a maximum reaction time t. Using BeSpaceD, we verify in step 5 that the system fulfills the safety properties if t is kept, i.e., $S \wedge R(t) \Rightarrow P$. In step 6, we prove with UPPAAL that the system guarantees the maximum reaction time, i.e., $S \Rightarrow R(t)$. It is evident that the combination of the two proofs implies $S \Rightarrow P$ such that the Reactive Blocks model created in step 2 effectively fulfills the spatial properties defined in step 1.

Further, we have to argue whether our model is indeed a correct abstraction of the real physical system in which the generated code of the controller shall be used. In particular, it is important to understand if and under which conditions the real system may violate P even if the two proof steps succeed. Preserving safety properties throughout refinement and reuse of verification results achieved on abstract models has been studied in the past, e.g., (Loiseaux, Graf, Sifakis, Bouajjani, & Bensalem, 1995) and is especially important in the context of model checking. When regarding space, we distinguish here between the following:

- **Overapproximation of spatial behavior:** For instance, the size of a spatial area occupied by a unit can be extended for proving the absence of collisions. This enables us to reuse previous spatial proofs if the sizes of a physical model do not exceed the overapproximated ones taken as a basis in the former proof;
- **Underapproximation of spatial behavior:** Just as the overapproximations, we can, for example, underapproximate sensor ranges that allow the detection of other units, such that established properties can be reused in later development stages;

- **Overapproximation of maximum reaction times:** The BeSpaceD verifications in step 5 are valid if we assume greater or equal reaction times than those guaranteed by the controller, the sensors and actuators in the worst case.

1.1. Guiding Example: Moving Robot

We intend to protect maintenance personnel in a factory hall against collisions with a fast moving robot transporting goods. The layout of the scenario is depicted in Figure 1. The robot has a size of up to 2 x 2 meters and moves on a straight line in the center of the room covering a distance of 100 *m*. Since it reaches a speed of up to 10 *m*/*s*, a collision with a human may lead to fatal injuries. The hall is equipped with sensors observing if a human approaches the robot and we shall design a safety controller which may stop the robot in due time. The robot can be operated in three different modes depending on its distance to humans in the hall:

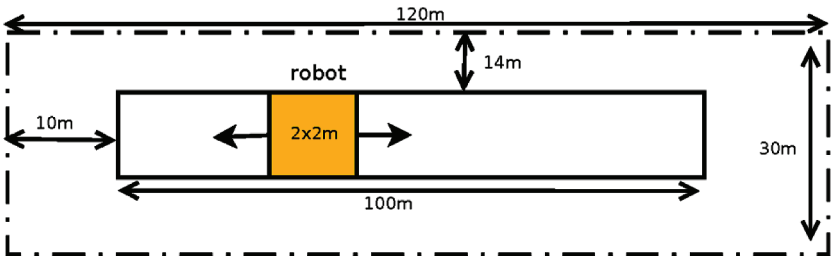
- **Normal mode:** If no human is closer than 25 *m* to the robot, the robot increases its speed with an acceleration of 5 m/s^2 until reaching 10 *m*/*s*. After 87 *m*, the speed of the robot is reduced with an acceleration of -5 m/s^2 until it amounts to 1 *m*/*s* which is kept until the robot reaches the buffer stop at the endpoint;
- **Amber mode:** If a human is detected in a distance of less than 25 *m* but more than 10 *m*, the robot is slowed down with an acceleration of -10 m/s^2 until reaching a speed of 2 *m*/*s* (resp. 1 *m*/*s* if more than 87 *m* are passed);
- **Red mode:** If the distance between the robot and a human is less than 10 *m*, the robot is stopped with an acceleration of -15 m/s^2.

For the mode changes, we assume a latency of 500 *ms* at maximum reflecting that there will be communication delays between the physical components as well as processing times of the robot controller. Of course, the main spatial proof task is to find out if the selected behavior of the safety control ensures that the robot is sufficiently slow or already stopped when a human reaches it. An aggravating fact is that the sensor may detect humans only if they are in the hall but that the face side is only 10 *m* and the long side 14 *m* from the path of the robot.

1.2. Synopsis

Steps 1 and 2 of the workflow, in particular, the engineering of controllers with Reactive Blocks will be outlined in Section 2. In Section 3, step 5, i.e., the proof of spatial properties with BeSpaceD, is discussed followed by an introduction of step 4, the coupling between Reactive Blocks and BeSpaceD, in Section 4. The use of UPPAAL to verify real-time properties in step 6 is described in Section 5. The paper is concluded with a discussion of related work and a conclusion.

Figure 1. Layout of the moving robot



2. MODELING CONTROLLERS AND CONTINUOUS BEHAVIOR

The input of BeSpaceD can be either direct logical terms or a Scala program generating the terms. For the moving robot example, we started the workflow with a Scala program modeling the spatial behavior of the robot and the human. Here, for instance the speed that the robot shall carry in the normal mode can be described by the code snippet shown in Algorithm 1.

For step 2, we apply the engineering tool Reactive Blocks (Kraemer, Slåtten, & Herrmann, 2009) that allows the model-based development of reactive systems all the way from abstract behavioral specifications to executable code. It enables to specify subfunctions of an application in separate models that we call building blocks. Using synchronous coupling, building blocks can be further composed to system models. An advantage of this proceeding is that a functionality recurring in several applications is specified once as a building block that can be reused in various models (Kraemer & Herrmann, 2009). The behavior of a building block is specified as a UML 2 activity while its behavioral interface is described by a so-called External State Machine (ESM) (Kraemer & Herrmann, 2009). Reactive Blocks uses a formal semantics (Kraemer & Herrmann, 2010) such that in step 3 of our workflow the activities can be model checked for design properties, e.g., compliance of an activity with the ESM of its building block (Kraemer, Slåtten, & Herrmann, 2009). System models can be automatically transformed to executable Java code (Kraemer & Herrmann, 2007), (Kraemer, Herrmann, & Bræk, 2006) which corresponds to step 7 of the workflow.

Figure 2 depicts a UML activity modeling the behavior of a building block. Similarly to Petri nets, behavior is expressed as tokens passing via the edges of a graph towards its vertices which may be flow control units like forks duplicating tokens or timers as well as operations containing Java methods (e.g., *computeMode*). Further, an activity may contain call behavior actions like *Timer Periodic 2* referring to other building blocks. The interaction of the activity describing the behavior of a building block *B* with the one including a call behavior action referring to *B*, is modeled by pins and parameter nodes. The parameter nodes are described as little squares at the edges of the activity, e.g., *new1* in Figure 2, while the pins are similar symbols at the edges of the call behavior actions (e.g., *tick* in *Timer Periodic 2*). The pins at a call behavior action of a building block are identical to the parameter nodes of its activity.

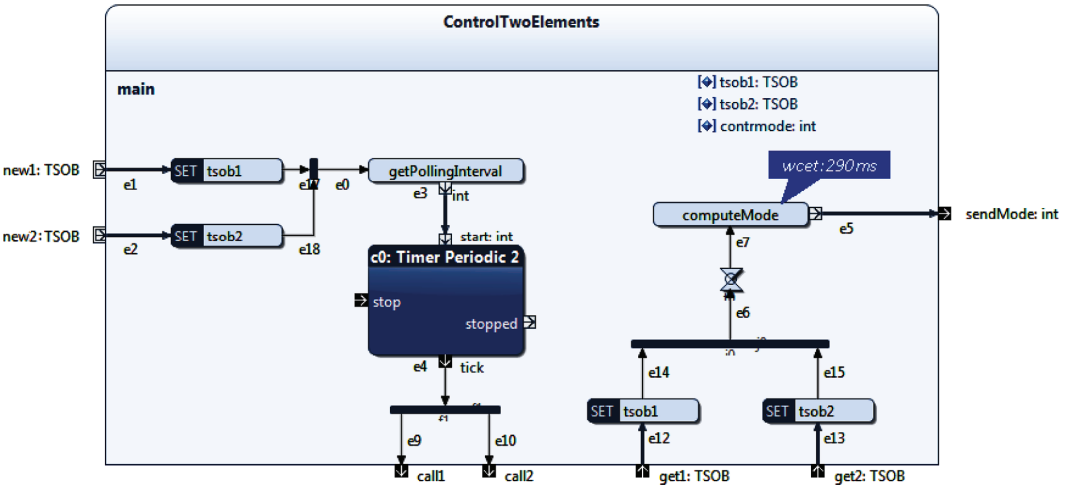
In Reactive Blocks, we follow run-to-completion semantics modeling that tokens flow via several edges and nodes in a single atomic step, a so-called *activity step*, until they have to wait for other behaviors like receiving an event from another station or a timeout (Kraemer & Herrmann, 2010). An activity step may encompass several activities since the "hop" of a token between two activities via pins and parameter nodes is carried out synchronously.

The UML activity in Figure 2 models the behavior of the building block *ControlTwoElements* that we use to specify the safety controller in our example. It is a feedback controller that polls the sensors of the the robot as well as the human and uses the sensor data to compute the correct control mode of the robot. The activity is initiated by two simultaneously arriving data tokens via the parameter nodes *new1* and *new2* containing location information about the robot and the human at system start. The corresponding time, location and speed data is defined by the Java class *TSOB* which is the type of

Algorithm 1. Code snippet

```
if (mode == 3) { // normal
  if (d <= 870) {
    if (speed < 10) {
      speed += 0.0005 * timeraster; } }
  else {
    if (speed > 1) {
      speed -= 0.0005 * timeraster; } } }
```

Figure 2. UML activity of building block ControlTwoElements



both parameter nodes. In the same activity step, the two data tokens are stored in the variables *tsob1* and *tsob2* and the two tokens are joined to one passing operation *getPollingInterval*. This operation refers to a Java method that reads out a parameter of the building block which describes the time interval between two polls of the sensors (5 *ms* in the example). Thereafter, the token is forwarded starting the building block *Timer Periodic 2* that will periodically issue timeouts according to the value of the parameter.

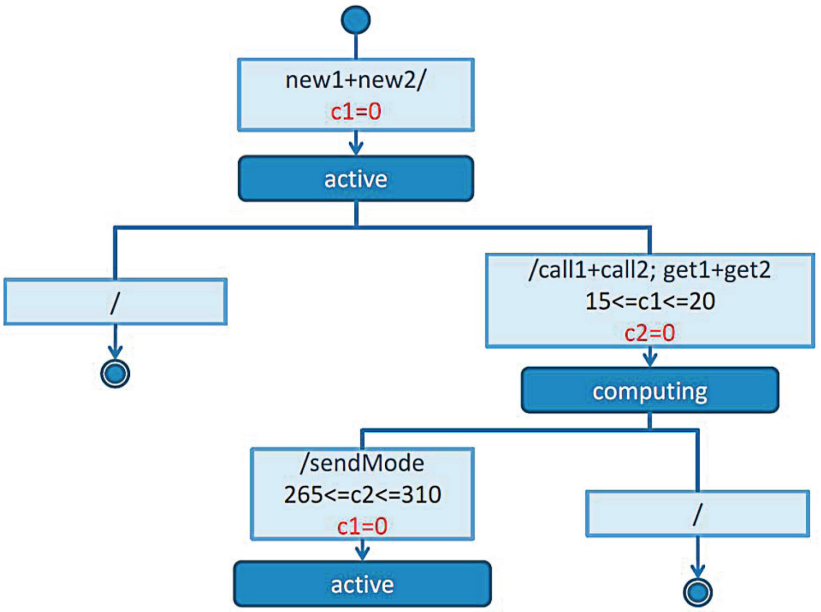
A timeout is modeled by a token arriving through pin *tick* of block *Timer Periodic 2* that is duplicated in the succeeding fork and outputted via the parameter nodes *call1* and *call2*. Assuming that the drivers of the sensors rest in the same physical component as the controller, we can model that the sensor data arrives still in the same activity step through the pins *get1* and *get2*. The data units are stored in the variables *tsob1* and *tsob2* followed by a join towards a flow breaker. That is a timer without a waiting time which sole purpose is to separate two activity steps.

In a new activity step, the token leaves the flow breaker and causes the execution of the Java method *computeMode* which takes the sensor data from the variables *tsob1* and *tsob2* and computes in which of the modes *normal*, *amber* or *red* the robot has to operate. The mode values are typified by integer values that are outputted via parameter node *sendMode*.

In Figure 3, we point out the ESM of building block *ControlTwoElements*. The markings[2] at the edges of the state machine refer to the parameter nodes and model which parameter nodes are passed by tokens in a certain activity step. For instance, in the activity step leading from the initial node of the ESM to state *active*, tokens pass through both parameter nodes *new1* and *new2*. In state *active*, a transition may be executed which leads towards ESM state *computing*. It models the polling of sensor data and refers to the parameter nodes *call1* and *call2* followed by an immediate reaction via *get1* and *get2*. In state *computing*, a transition consisting of a flow via parameter node *sendMode* is allowed reflecting the transmission of new control modes. This transition sets the ESM back to state *active*. By the symbol */* in the transition markings one expresses if a transition is triggered by the activity in a building block or by its environment.

The overall system consists of twelve building blocks and, due to the similarity with Java, we profitted from the Scala code created in step 1 of the workflow. For instance, the code snippet shown above was copied and pasted into an operation of building block *ContinuousStepRobot* that specifies the robot behavior in the Reactive Blocks model.

Figure 3. The RTESM (ESM) of building block ControlTwoElements



3. PROVING SPATIAL PROPERTIES WITH BESPACED

We implemented BeSpaceD (Blech & Schmidt, 2013), a tool for checking spatial behavior of cyber-physical systems, in the programming language Scala. The description language of BeSpaceD allows to define abstract datatypes that indicate spatial availability, interaction or occupation in areas in a coordinate system for time intervals or timepoints. As mentioned above, we can also use Scala code to generate BeSpaceD specifications. The language allows to specify physical system behavior on various abstraction levels reaching from simple models regarding only distinct availability areas at certain time intervals or timepoints to complete behavioral models. In particular, we can describe the space covered by a system at a certain timepoint in form of rectangles and other shapes. Further, we can constrain the coordination of different systems by allowing interaction only if their locations are within a certain distance. To give a look and feel, the code segment shown in Algorithm 2 points out an extract of the moving robot behavior.

Algorithm 2. An extract of the moving robot behavior

```
AND(AND (AND (
  IMPLIES (TimeStamp (410),
    OccupyBox (160, 139, 182, 161)),
  IMPLIES (TimeStamp (411),
    OccupyBox (161, 139, 183, 161))),
  IMPLIES (TimeStamp (412),
    OccupyBox (161, 139, 183, 161))),
  IMPLIES (TimeStamp (413),
    OccupyBox (162, 139, 184, 161)));
```

The logical formula expresses that the robot covers a rectangle defined by the corner points (160 x 139) and (182 x 161) at the timepoint 410 which describe coordinates in the hall in decimeters. At the next timepoint the robot moved a decimeter to the right covering the box between (161 x 139) and (183 x 161) etc.

Spatial verification with BeSpaceD can be difficult, if moving objects show excessive non-deterministic behavior since that multiplies the scenarios to be checked. In the example, we are able to specify the robot behavior exactly and use it for verification purposes. The human, however, can freely change speed and direction showing a high degree of non-deterministic behavior. Two solutions exist to cope with non-determinism: We can describe the non-deterministic behavior in a more abstract way, such that the number of verification scenarios is reduced. For instance, we can express the behavior of humans by rectangles describing all places, they may have reached at a certain timepoint. This may have the disadvantage, that scenarios are too coarse-grained, so that important safety properties cannot be verified even if they hold in reality, e.g., the rectangles modeling the possible position of a human will gradually grow until they cover the whole factory hall. Alternatively, we can select a set of worst-case scenarios and check only those. For a given case study, however, we have to argue why the chosen set of worst-case scenarios is indeed sufficient. No general solution exists so far for the automatic selection of worst-case scenarios.

For the most important proof of step 5 in our workflow, i.e., the robot runs only very slowly or already stands when it is reached by a human, we chose the second solution. Due to the basic laws of kinematics, we could restrict us to two worst-case scenarios, i.e., running from the door on the face side of the room (see Figure 1) against the moving direction of the robot, resp. entering the room through the door on the long side and running in a right angle towards its path. In these scenarios, we considered both, the highest possible approximation speed between robot and human and the situations in which the human is closest to the robot when the sensor detects an approximation. Since the robot needs 12.6 *s* for the overall run if it is in normal mode and we assume 5 *ms* between two timepoints, its behavior can be described by altogether 2520 timepoints. To be sure that by discretizing the robot behavior we did not overlook unsafe situations, we overapproximated the rectangle of the robot and assumed that it covers 2.2 x 2.2 *m*. As the human may enter the room any time, we created for each of the two scenarios 2520 variants such that the human may start its run at any of the timepoints defining the current robot location.

In the verification process, specifications realizing a verification scenario are given in the BeSpaceD language and are broken down to expressions containing geometrical information. Yet another automatic transformation breaks these geometric expressions down to representations that are suitable for solving algorithms and special solvers like SMT[3] and SAT[4]. Further, we use a simpler and faster hashset-based implementation for a subclass of SAT problems that checks possible collisions between two entities, each one defined by multiple points in space and time. The overall 5040 runs of the two scenarios could be proven by this refined prover within five minutes on a standard PC.

For the human, we assumed a maximum speed of 10 *m*/*s* causing a relative speed of up to 20 *m*/*s* if the user enters the room from the face side. In spite of that and a distance of only 9.8 *m* between the door and the final position of the robot, this scenario was verified as safe for all starting points of the human since the robot already stands at the time of impact. The reason for that is that the robot is either sufficiently far from the human when the latter enters the room such that the controller has enough time to react, or it is already slowing down approaching its final point such that the breaking distance is shorter.

The problem in the other scenario is that the robot may be in full speed while the human enters the room from the side in a distance as close as 13.8 *m*. Indeed, if the human enters the room when the robot is about 9.5 *m* before the point of impact, it can be reached before having completely stopped. Simulating this case with the program generated from Reactive Blocks, we found out that the maximum speed of the robot at the time of impact is at most 0.625 *m*/*s*. It has to be decided if the risk of such an encounter, which is unlikely since the speed of 10 *m*/*s* can only be reached by

few athletes and depending on the physics of the robot might not cause severe injuries as the robot is nearly standing at the time of impact, can be beard or if the control software resp. the environment have to be changed.

4. COMPOSING REACTIVE BLOCKS AND BESPACED

In our first experiment (Han, Blech, Herrmann, & Schmidt, 2014), we used simulator runs to achieve step 4 of the workflow coupling Reactive Blocks with BeSpaceD: The Reactive Blocks model of the simulator was amended by operations writing the positions of robot and human at each time point as an IMPLIES statement into files that formed the input for BeSpaceD. An apparent disadvantage of this coupling method is that the need to perform simulator runs for all scenarios can be rather time-consuming. So, assuming 12.6 seconds for the run of each of the altogether 5040 variants to be checked in the two scenarios (see Section 3), the simulator has to be executed nearly 18 hours just to generate the BeSpaceD input data. Further, by this kind of integration we limit ourselves to purely scenario-based proofs in BeSpaceD which does not live up to the capabilities of this tool.

As a consequence, we need a design time-based coupling between the two tools not necessitating simulator runs at all. The basic idea is to exploit the ability of BeSpaceD to use Scala files as input and to copy the Java code that in a Reactive Blocks model realizes the controller and simulator, directly into the corresponding Scala files. As already discussed in Section 2, compiling Java into Scala is merely trivial since the syntaxes of the operations in both languages are identical and only the variable declarations have to be adjusted by the transformation tool.

The main problem of an automatic transformation from Reactive Blocks to BeSpaceD is to find out where the relevant code segments reside in an arbitrary Reactive Blocks model. To solve this, we utilize the property mentioned in Section 2 that Reactive Blocks models can be composed from reusable building blocks (Kraemer & Herrmann, 2009). We created a new library of building blocks for the domain of cyber-physical systems. That does not only help to create the controller and simulator models in step 2 of our workflow but can also be used as starting point for graphical pattern detection in order to find code segments to be transformed to BeSpaceD. Up to now, the library contains building blocks for several types of controllers, a block *ContinuousStep* to create various simulators, and *TimeStampOccupyBoxManager* managing the instances of type *TSOB* containing spatiotemporal information that can directly be handled by BeSpaceD.

The transformation tool searches a Reactive Blocks model for the occurrence of these blocks. This is basically achieved in three steps:

1. An instance of block *TimeStampOccupyBoxManager* contains the interval between two time points as a parameter that can be directly used in the Scala input file of BeSpaceD to compute the intervals between two timepoints;
2. The movement and speed of a physical unit depending on its current execution mode (e.g., *normal*, *amber* or *red* in the robot system) are computed by an operation in the direct environment of block *ContinuousStep* that can be retrieved by the transformation tool and copied into the Scala program;
3. The execution mode of a unit is computed in a certain operation of its controller block. For example, in building block *ControlTwoElements* (see Figure 2) that is operation *computeMode*, the content of which is also copied into the Scala program.

The first and second transformation steps are carried out for each physical component of the system and the third one for each controller component used.

Of course, to guarantee the correctness of the transformations, the composition of the building blocks has to fulfill certain properties: For instance, the interval between two simulator steps has to be in

accordance with the parameter defined in block *TimeStepOccupyBoxManager*. Otherwise, BeSpaceD would possibly use a wrong assignment between time stamps and positions of the geographical objects. Further, an operation mode computed in a controller block must be directly forwarded to block *ContinuousStep*. This is analyzed by the transformation tool using graph transformation techniques (Han & Herrmann, 2012).

5. VERIFYING REAL-TIME PROPERTIES

To achieve step 6 of our workflow, we extend the interface descriptions by using Real-Time ESMs (RTESMs) (Han & Herrmann, 2013), (Han, Herrmann, & Le, 2013) that allow to specify that a building block may only rest in an RTESM state for a maximum period of time before a transition has to be fired. RTESMs extend the ESMs with time variables, so-called clocks, as well as a set of labels expressing clock reset, state invariants and guard conditions. In the RTESM of building block *ControlTwoElements* in Figure 3, the state invariants are marked in black and the clock resets in red. Moreover, one can annotate the various vertices of an activity by worst case execution time attributes (Han & Herrmann, 2013). For instance, in the activity of block *ControlTwoElements* we assigned 290 *ms* to operation *computeMode* (see annotation in Figure 2). For some activity vertices, we further assume certain default delays, e.g., 2 *ms* for writing and reading variables as well as for the start of a new activity step.

As explained in (Han & Herrmann, 2013), (Han, Herrmann, & Le, 2013), the RTESM and the activity of a building block are automatically transformed to Timed Automata (Alur & Dill, 1990) in which the real-time annotations introduced above are considered. Thus, one can use the model checker UPPAAL (Bengtsson, et al., 1996) to prove timed properties expressed in Timed-CTL (TCTL) (Laroussinie, Markey, & Schnoebelen, 2004) and to verify whether the activity indeed fulfills the RTESM of its block. The TCTL formulas are also automatically generated.

The approach is highlighted with our moving robot example. Table 1 describes the minimum resp. maximum execution times that we assume for the various tasks of the control cycle in our example. The sum of the worst case execution times (*wcet*) of all four steps is exactly the 500 *ms* assumed in the BeSpaceD proof in Section 3. The transition from state *active* to state *computing* in the RTESM models the fetching of sensor data which includes the time between two polling calls. We expect that this task is handled within 15 to 20 *ms* which is expressed by clock *c1* in the RTESM enforcing that the transition is executed within this time interval. For the latency in state *computing*, we assume between 265 and 310 *ms* which corresponds to the sum of the image processing time in the controller and the communication delay towards the robot controller. It is modeled by means of clock *c2*. In a similar way, we defined the RTESMs of the other building blocks forming our example system. The operation *computeMode* in the activity of block *ControlTwoElements* (Figure 2) is annotated with a *wcet* of 290 *ms* since it contains the code to process the execution mode from the sensor inputs. Likewise, we annotated the other activities by suitable *wcet* attributes.

A critical element for the proof that building block *ControlTwoElements* fulfills its RTESM, is the flow breaker ahead of operation *computeMode*. During the execution of the activity step leading

Table 1. Maximum and minimum execution times of different tasks of the robot control system

Component	Min. Time	Max. Time
Time to fetch sensor data including polling delay	15 *ms*	20 *ms*
Processing time recognition unit	250 *ms*	290 *ms*
Communication time recognition unit to robot	15 *ms*	20 *ms*
Internal robot processing time and actuator reaction	150 *ms*	170 *ms*

towards the flow breaker, other activity steps may be added to the execution queue such that they are carried out earlier than the one leaving the flow breaker. Considering the time guarantees of the environment as defined in the RTESM and the *wcets* of the other activity steps, the waiting time until the activity step being triggered from the flow breaker is at most 8 *ms*. Thus, together with the *wcet* of 290 *ms* defined for operation *computeMode*, the building block will stay at most 298 *ms* in its RTESM state *computing* which is lower than the 310 *ms* guaranteed by the RTESM.

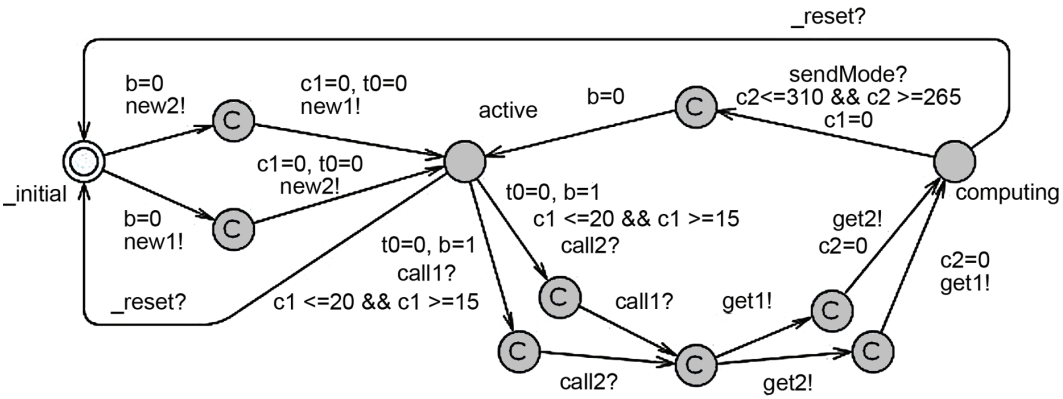
The Timed Automaton transformed from the RTESM of block ControlTwoElements is depicted in Figure 4. For the communication between a Timed Automaton representing an RTESM and the one of the activities of its building block resp. its environment, we use synchronization channels (e.g., *new1*). Here, RTESM transitions with multiple parameters are expressed by interleaving (e.g., from state *_initial* to *active* in Figure 4). UPPAAL can now use the Timed Automata to prove whether the real-time properties are fulfilled. For instance, the proof discussed above is carried out by checking if the Timed Automaton of the RTESM receives signal *sendMode* from the one modeling the activity within 310 *ms* guaranteeing that the state invariant of this state is fulfilled. This corresponds to proving the TCTL formula $A[]$ (*external.computing imply* $c2 \leq 310$). Altogether, we executed 22 UPPAAL proofs which were completed within some milliseconds each. This is due to a relatively small number of states in all Timed Automata effectively exploiting the compositional structure of the Reactive Blocks models (see Kraemer et al., 2009).

6. RELATED WORK

Work relevant to this paper has been done in areas such as formal logic and process algebras, hybrid-systems, robotics and formal methods for component-based software engineering.

Duration Calculus (Chaochen, Hoare, & Ravn, 1991) and timed Durational Action Timed Automata (Guellati, Kitouni, & Saidouni, 2012) are two types of modal formalisms for time-critical systems. They are especially used for analyzing parallel behavior of systems featuring actions with an elapsing non-atomic time duration. Complementing classical modeling approaches, for the specification of system models comprising spatial behavioral information, a process algebra-like formalism was introduced in (Caires & Cardelli, 2003), (Caires & Cardelli, 2004). Here, disjoint logical spaces are represented in terms of expressions by bracketing structures and carry or exchange concurrent process representations. For additional results on spatial interpretations see, e.g., (Hirschkoff, Lozes, & Sangiorgi, 2003). Many aspects of spatial logic are in general undecidable. A quantifier-free rational fragment of ambient logic (corresponding to regular language constraints), however, was shown to be decidable in (Dal Zilio, Lugiez, & Meyssonnier, 2004). Special modal logics for spatiotemporal

Figure 4. Timed automata of RTESM for block ControlTwoElement



reasoning go back to the seventies. The Region Connection Calculus (RCC) (Bennett, Cohn, Wolter, & Zakharyaschev, 2002) includes spatial predicates of separation (e.g, regions do not share points at all, points on the boundary of regions are shared, proper overlap of regions, or proper inclusion). Moreover, (Bennett, Cohn, Wolter, & Zakharyaschev, 2002) features an overview of the relation of these logics to various Kripke-style modal logics, reductions of RCC-style fragments to a minimal number of topological predicates, their relationship to interval-temporal logics and decidability.

The area of hybrid systems has seen the development of different tools for reasoning and verification. SpaceEx (Frehse, et al., 379-395) allows the modeling of continuous hybrid systems based on hybrid automata. It can be used for computing overapproximations of the space occupied by an object moving in time and space. In addition, it is possible to model spatial behavior in more general purpose-oriented verification tools in Hybrid systems, e.g., (Platzer & Quesel, 2008). Formal methods have also been widely used for safety property analysis in safety-critical systems (Slåtten, Herrmann, & Kraemer, 2013). E.g., in (Németh & Bartha, 2009), the authors use a CTL-subset for the verification of a safety procedure called primary-to-secondary leaking (PRISE) that is discussed by means of a model of a nuclear power plant.

7. CONCLUSION

We introduced a tool chain for the formal-based engineering of controllers for embedded systems that have to fulfill certain spatial behavioral properties. The models are created with Reactive Blocks while the verification is carried out with BeSpaceD and UPPAAL. In particular, we developed a capable and highly automatic transformation mechanism between Reactive Blocks and BeSpaceD. Further, we created a library of reusable building blocks supporting the pattern matching used by this tool coupling. The building blocks of this library help also to create new Reactive Blocks models. In the adapted model of the example four of the overall twelve building blocks were copied from this library while further two were taken from another library of Reactive Blocks. Confirming our experience described in (Kraemer & Herrmann, 2009), the new model could be created in about a third of the time needed for the first one presented in (Han, Blech, Herrmann, & Schmidt, 2014) for which the library did not exist yet. Except for the reason, that we needed only to develop six of the twelve blocks, we profited also from the interface layout of the blocks which provided a good guidance for the planning of the control and information flows in the model.

Besides extending the analysis tool of Reactive Blocks (Kraemer, Slåtten, & Herrmann, 2009) to carry out the property proofs discussed in Section 4, in the future, we plan to investigate additional specification mechanisms, i.e., spatial behavioral types following (Blech & Schmidt, 2013) for spatial verification aiming at storing, composing and reusing verification results. Moreover, we are interested in further optimizations and parallelization of the verification process using cloud and grid technology to speed up the analysis process of BeSpaceD and the model checkers. For instance, it should not be a problem to parallelize the BeSpaceD runs of the 5040 variants that we had to verify for the two worst-case scenarios. As an application domain for our work, we see the emerging field of Wireless Sensor Networks (WSNs). In particular, it seems highly interesting and practically relevant to find optimal correlations between spatial properties, communication channel bandwidth and signalling strength of the WSN transmitters and our approach might be helpful to solve this problem.

REFERENCES

Alur, R., & Dill, D. (1990). Automata for Modeling Real-Time Systems. *Automata, Languages and Programming, LNCS*, *443*, 322–335.

Bengtsson, J., Larsson, F., Pettersson, P., Yi, W., Christensen, P., & Jensen, J. et al. (1996). UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems. *Hybrid Systems III, LNCS*, *1066*, 232–243.

Bennett, B., Cohn, A., Wolter, F., & Zakharyaschev, M. (2002). Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. *Applied Intelligence*, *17*(3), 239–251. doi:10.1023/A:1020083231504

Blech, J., & Schmidt, H. (2013). Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems. *Improving Systems and Software Engineering Conference (ISSEC)*. Melbourne.

Caires, L., & Cardelli, L. (2003). A Spatial Logic for Concurrency (Part I). *Information and Computation*, *186*(2), 194–235. doi:10.1016/S0890-5401(03)00137-8

Caires, L., & Cardelli, L. (2004). A Spatial Logic for Concurrency (Part II). *Theoretical Computer Science*, *322*(3), 517–565. doi:10.1016/j.tcs.2003.10.041

Chaochen, Z., Hoare, C., & Ravn, A. (1991). A Calculus of Durations. *Information Processing Letters*, *40*(5), 269–276. doi:10.1016/0020-0190(91)90122-X

Dal Zilio, S., Lugiez, D., & Meyssonnier, C. (2004). A Logic You Can Count On. *Symposium on Principles of Programming languages*. ACM.

De Moura, L., & Bjørner, N. (2008). An Efficient SMT Solver. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 4963, 337-340.

Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., . . . (2011). SpaceEx: Scalable Verification of Hybrid Systems. Computer Aided Verification (CAV), LNCS 6806, 379-395.

Guellati, S., Kitouni, I., & Saidouni, D. (2012). Verification of Durational Action Timed Automata using UPPAAL. *International Journal of Computers and Applications*, *56*(11), 33–41. doi:10.5120/8938-3077

Han, F., Blech, J., Herrmann, P., & Schmidt, H. (2014). *Towards Verifying Safety Properties of Real-Time Probability Systems*. Electronic Proceedings in Theoretical Computer Science.

Han, F., & Herrmann, P. (2012). Remedy of Mixed Initiative Conflicts in Model-based System Engineering. *Electronic Communications of the EASST, 47*.

Han, F., & Herrmann, P. (2013). Modeling Real-Time System Performance with Respect to Scheduling Analysis. *Proceedings of the 6th IEEE International Conference on Ubi-Media Computing* (pp. 663-671). IEEE Computer. doi:10.1109/ICAwST.2013.6765522

Han, F., Herrmann, P., & Le, H. (2013). Modeling and Verifying Real-Time Properties of Reactive Systems. *18th International Conference on Engineering of Complex Computer Systems (ICECCS)* (pp. 14-23). IEEE Computer. doi:10.1109/ICECCS.2013.13

Hirschkoff, D., Lozes, É., & Sangiorgi, D. (2003). Minimality Results for the Spatial Logics. *Foundations of Software Technology and Theoretical Computer Science, LNCS 2914.*

Kraemer, F., & Herrmann, P. (2007). Transforming Collaborative Service Specifications into Efficiently Executable State Machines. *Electronic Communications of the EASST, 7*.

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

Kraemer, F., & Herrmann, P. (2010). Reactive Semantics for Distributed UML Activities. *Joint WG6.1 International Conference (FMOODS) and WG6.1 International Conference (FORTE), LNCS 6117* (pp. 17-31).

Kraemer, F., Herrmann, P., & Bræk, R. (2006). Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. *Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA06), LNCS 4276* (pp. 1613-1632). Springer-Verlag. doi:10.1007/11914952_41

Kraemer, F., Slåtten, V., & Herrmann, P. (2009). Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software*, *82*(12), 2068–2080. doi:10.1016/j.jss.2009.06.057

Laroussinie, F., Markey, N., & Schnoebelen, P. (2004). Model Checking Timed Automata with One or Two Clocks. *Concurrency Theory (CONCUR), LNCS*, *3170*, 387–401.

Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S., & Probst, D. (1995). Property Preserving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design*, *6*(1), 1–35. doi:10.1007/BF01384313

Németh, E., & Bartha, T. (2009). Formal Verification of Safety Functions by Reinterpretation of Functional Block Based Specifications. *Formal Methods for Industrial Critical Systems, LNCS 5596*, 199–214.

Platzer, A., & Quesel, J. (2008). KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). *Automated Reasoning, LNCS 5195*, 171–178.

Slåtten, V., Herrmann, P., & Kraemer, F. (2013). Model-Driven Engineering of Reliable Fault-Tolerant Systems - A State-of-the-Art Survey. *Advances in Computers*, *91*, 119–205. doi:10.1016/B978-0-12-408089-8.00004-5

Tyagi, A. K. (2012). *MATLAB and Simulink for Engineers*. Oxford University Press.

ENDNOTES

1 Until recently, Reactive Blocks was named Arctis.
2 Ignore for the moment the black and red real-time extensions in the markings.
3 We implemented a transformation to Z3 (De Moura & Bjørner, 2008).
4 We implemented a transformation to Sat4j: http://www.sat4j.org/.

Call for Articles

International Journal of Web Services Research

Volume 13 • Issue 1 • January-March 2016 • ISSN: 1545-7362 • eISSN: 1546-5004
An official publication of the Information Resources Management Association

MISSION

Web Services are among the most important emerging technologies in the e-business, computer software and communication industries. The Web Services technologies will redefine the way that companies do business and exchange information in twenty-first century. They will enhance business efficiency by enabling dynamic provisioning of resources from a pool of distributed resources. Due to the importance of the field, there is a significant amount of ongoing research in the areas. In a parallel effort, standardization organizations are actively developing standards for Web Services. The Web Services are creating what will become one of the most significant industries of the new century. The **International Journal of Web Services Research (IJWSR)** is designed to be a valuable resource providing leading technologies, development, ideas, and trends to an international readership of researchers and engineers in the field of Web Services.

COVERAGE AND MAJOR TOPICS

The topics of interest in this journal include, but are not limited to:

Business grid • Business process integration and management using Web services • Case studies for Web services • Communication applications using Web services • Composite Web service creation and enabling infrastructures • Dynamic invocation mechanisms for Web services • E-commerce applications using Web services • Frameworks for building Web service applications • Grid-based Web services applications (e.g. OGSA) • Interactive TV applications using Web services • Mathematic foundations for service oriented computing • Multimedia applications using Web services • Quality of service for Web services • Resource management for Web services • Semantic services computing • SOAP enhancements • Solution management for Web services • UDDI enhancements • Web services architecture • Web services discovery • Web services modeling • Web services performance • Web services security

ALL INQUIRIES REGARDING IJWSR SHOULD BE DIRECTED TO THE ATTENTION OF:

Liang-Jie Zhang, Editor-in-Chief • IJWSR@igi-global.com

ALL MANUSCRIPT SUBMISSIONS TO IJWSR SHOULD BE SENT THROUGH THE ONLINE SUBMISSION SYSTEM:

http://www.igi-global.com/authorseditors/titlesubmission/newproject.aspx

IDEAS FOR SPECIAL THEME ISSUES MAY BE SUBMITTED TO THE EDITOR(S)-IN-CHIEF

PLEASE RECOMMEND THIS PUBLICATION TO YOUR LIBRARIAN

For a convenient easy-to-use library recommendation form, please visit:
http://www.igi-global.com/IJWSR