Towards Verifying Safety Properties of Real-Time Probabilistic Systems

Fenglin Han
Norwegian University of Science and Technology, Trondheim, Norway
sih@item.ntnu.no

Jan Olaf Blech
RMIT University, Melbourne, Australia
janolaf.blech@rmit.edu.au

Peter Herrmann
Norwegian University of Science and Technology, Trondheim, Norway
herrmann@item.ntnu.no

Heinz Schmidt
RMIT University, Melbourne, Australia
heinz.schmidt@rmit.edu.au

Using probabilities in the formal-methods-based development of safety-critical software has quickened interests in academia and industry. We address this area by our model-driven engineering method for reactive systems SPACE and its tool-set Reactive Blocks that provide an extension to support the modeling and verification of real-time behaviors. The approach facilitates the composition of system models from reusable building blocks as well as the verification of functional and real-time properties and the automatic generation of Java code.

In this paper, we describe the extension of the tool-set to enable the modeling and verification of probabilistic real-time system behavior with the focus on spatial properties that ensure system safety. In particular, we incorporate descriptions of probabilistic behavior into our Reactive Blocks models and integrate the model checker PRISM which allows to verify that a real-time system satisfies certain safety properties with a given probability. Moreover, we consider the spatial implication of probabilistic system specifications by integrating the spatial verification tool BeSpaceD and give an automatic approach to translate system specifications to the input languages of PRISM and BeSpaceD. The approach is highlighted by an example.

1 Introduction

Modeling and verification methods for embedded control system in domains such as avionics, automotive and robotics should address a variety of software and hardware aspects including real-time and probabilistic properties, distribution of system components, communication protocols, characteristics of digital circuits and controllers. Real-time systems can require quantitative timing constraints which may include guaranteed probabilities for time and spatial properties. For example, a robot may be required to process a task in a predefined amount of time with a probability of 99.999999% to prevent expensive maintenance operations resulting from minor damage to the equipment. It must complete the task in a slightly larger amount of time with 100% to prevent major damage.

Here, we propose a framework for integrating probabilistic real-time verification and performance prediction with system development. This approach extends our existing model-driven engineering framework SPACE and its tool-set Reactive Blocks[1] [17] with real-time system behavior verification and schedulability analysis [12, 13]. Reactive Blocks enables the model-based engineering of reactive systems by composing reusable building blocks each describing a certain sub-functionality of a system.

[1] Until recently, Reactive Blocks was called *Arctis*.

B. Buhnova, L. Happe, J. Kofroň (Eds.): Formal Engineering Approaches to Software Components and Architectures 2014
EPTCS ??, 2014, pp. 1–15, doi:10.4204/EPTCS.??.??

The composed system model is automatically transformed into executable Java code [17]. Further, various formal verification methods ensure functional correctness [17] as well as reliability [27], security [9] and safety [12, 13] of targeted systems.

The formalism for modeling probabilistic real-time systems used in this work is based on probabilistic timed automata (PTA) [22]. It incorporates both probabilistic and real-time characteristics. Probabilistic properties are represented with an extension of Computational Tree Logic, i.e., PCTL [14]. PRISM [19] is a probabilistic model checker for formal analysis of systems that exhibits stochastic behaviors. It supports multiple-formalisms, including discrete-time Markov chains, continuous-time Markov chains, Markov decision processes and PTA making it possible to capture the random behavior of our real-time system model [12], for example random aspects of failure or uncertain inputs, loads or timing. We choose PTA for the stochastic behavior since it integrates well with our existing timed-automata based real-time system model. PTA has an equivalent descriptive power to a MDP (Markov Decision Process) [15] such that we can use the terminology of MDP for the system descriptions. Probabilistic CTL [18] has the capability of expressing real-time as well as probabilistic properties of a reactive system.

The tool set described in this paper involves the five engineering steps outlined below:

1. We model a system using Reactive Blocks including a simulator of its environment, in particular the spatial conditions to be reflected. In this model, we annotate probabilities as well as real-time behavior.

2. The model is analyzed with the model checker built into Reactive Blocks for functional errors and transformed into an executable simulator.

3. The simulator is carried out and, during the simulation, traces capturing spatiotemporal behavior with or without annotated probabilities are extracted for spatial verification.

4. The extracted spatiotemporal behavior is verified for possible spatial implications like collisions using our BeSpaceD tool [6]. Here, distributions capturing the combined probabilistic time behavior of the subcomponents are created from the extended Reactive Block models using the PRISM-based analysis.

5. If all analyses are passed, the simulator sub-functionality is removed from the Reactive Blocks model such that its core functionality can be transformed into executable code.

In this paper, we have three main contributions.

1. A novel approach for system performance predictions is introduced. In particular, we present a probabilistic real-time state-machine for software component performance descriptions. This so-called PRTESM is an extension of the External State Machines (ESMs) [16] used in Reactive Blocks. It allows to express probabilistic real-time assumptions and guarantees of a building block. That enables us to compose the PRTESMs of the various building blocks forming a system to predict probabilities of the overall system behavior.

2. We show the integration of the model-checker PRISM [19] to the tool-set. For that, the PRTESMs are transformed into PTAs [22] and the performance predictions of the overall system to PCTL statements [14] which can be directly proven by PRISM.

3. We look at spatial implications of probabilities in system behavior. The introduction of known probabilities into system behavior allows us to calculate how likely a physical unit like a robot will be present in a given area in space. We use the tool BeSpaceD [6] for this.

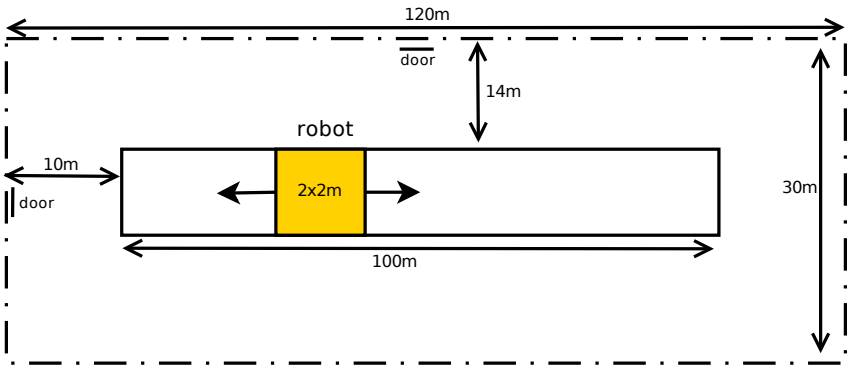


Figure 1: Layout of the moving robot

1.1 Guiding Example

We illustrate the tool set by a scenario of a moving robot in a factory hall featuring probabilistic behavior. Figure 1 provides a spatial layout of the example scenario. The moving robot occupies a 2 x 2 meters space in the 120 x 30 meters factory hall and moves along a straight line in the center of the room covering a distance of 100 meters. The maximum speed of the robot can reach 10 $\frac{m}{s}$ and thus a collision with a human may lead to fatal injuries. To eliminate such injuries, the hall is equipped with sensors monitoring the robot for approximations of humans. If the robot comes close to a human, it is slowed down or even stopped. The probabilistic aspect of this example comprises probability distributions on the reaction time once a human is detected. In this paper, the robot controller and a simulator of the continuous robot behavior are developed and implemented using Reactive Blocks.

1.2 Overview

The paper is arranged as follows: In Section 2, we give a description of the model of the robot control system example in Reactive Blocks realizing the distributed control functions as well as the robot simulation. Section 3 introduces our formalism for probabilistic time constraint and the translation into PRISM input. Section 4 presents our approach for spatial implications of probabilistic system behaviors and introduces the tool for probabilistic spatiotemporal property verification. We present the verification of our properties in Section 5. Related work is discussed in Section 6 followed by a conclusion in Section 7.

2 Modeling Control Functions

In the moving robot example sketched in Section 1.1, the three main activities are:

1. Polling of sensor data about the positions of the robot and the human[2].
2. Deciding the correct operation mode of the robot based on the distance to the human.
3. Forwarding an altered operation mode to the robot controller.

All three activities together should be performed within 0.5 s at maximum.

We model these control functions as well as the simulator of the human and robot behaviors separately from each other by different building blocks. In SPACE and Reactive Blocks, a building block

[2]For simplicity, we only consider the human closest to the robot.

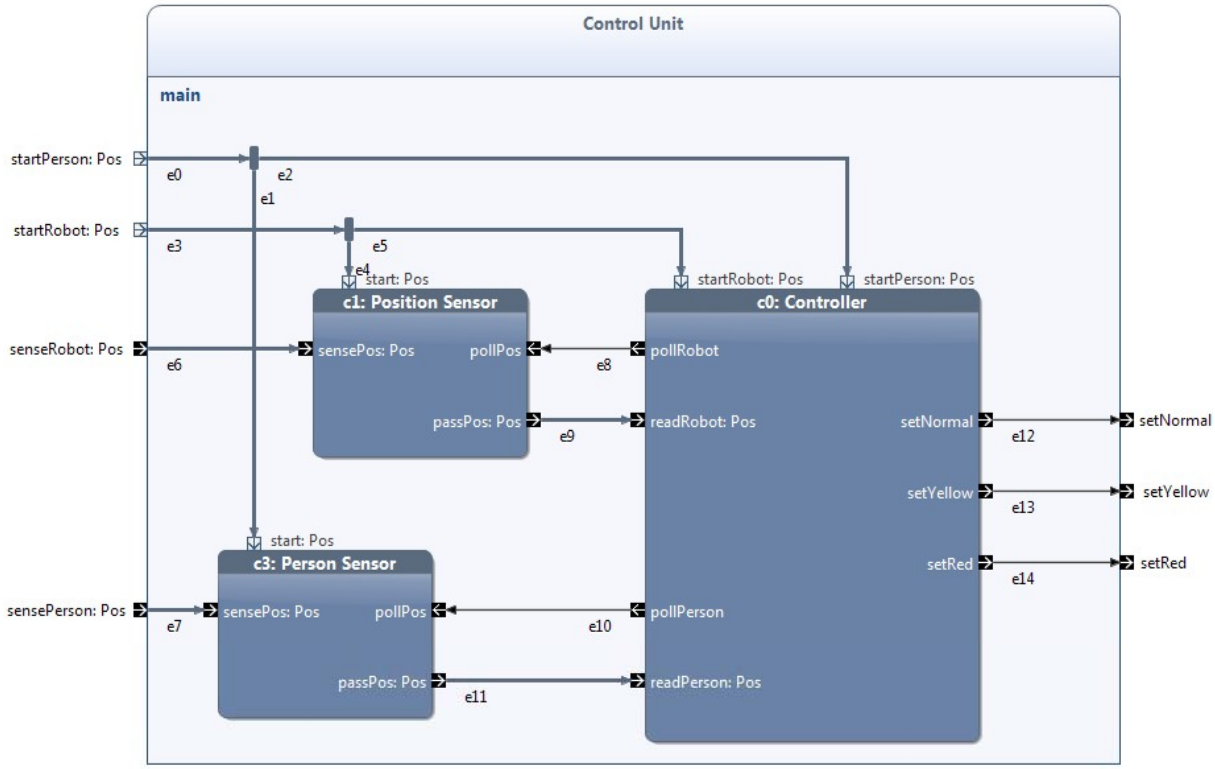
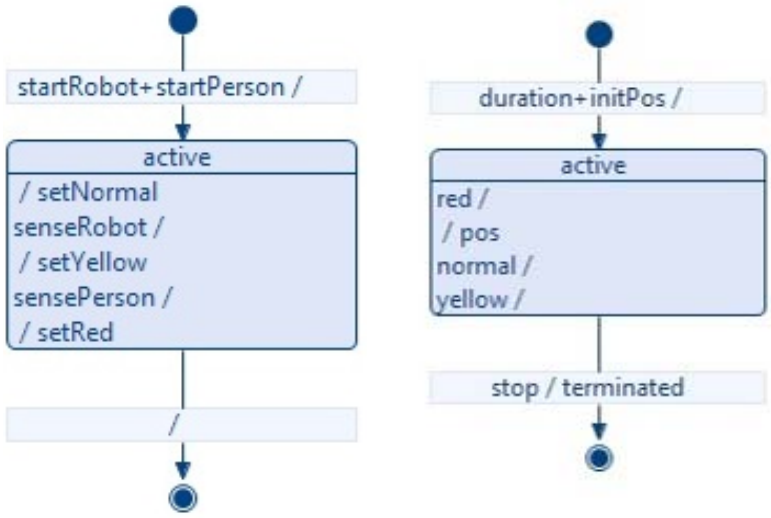


Figure 2: UML activity of building block *Control Unit*

consists of a behavioral model in the form of a UML activity [17] supplemented by an External State Machine (ESM) [16] describing its interface behavior.

The safety controller protecting humans from collisions with the fast moving robot is specified by the building block *Control Unit*. The UML activity modeling the behavior of this block is shown in Figure 2. Similar to Petri Nets, the control and data flows are represented by the flow of tokens in the activities. These tokens are passed by the activity edges towards vertices. Vertices can be control elements (such as forks duplicating tokens) or operations (associated with Java methods executed when a token arrives). Further, activities may contain call behavior actions like *Controller*, *Position Sensor* and *Person Sensor* each referring to another building block. The interaction between the activity containing a call behavior action of a building block and the one referring to its behavior is modeled by pins and parameter nodes. Parameter nodes are the identifiers at the edge of an activity, e.g., *startRobot* in block *Control Unit*. All parameter nodes of an activity are available as pins in the call behavior actions referring to its building block (e.g., *setRed* in block *Controller*). The semantics defines that a token reaching a pin of a call behavior action continues from the corresponding parameter node of the activity referring to the behavior of the call behavior action and vice versa.

Position Sensor and *Person Sensor* refer to the sensors for the positions of the robot and human which get the current position information from the simulation via their input pins *sensePos*. The block *Controller* realizes the safety controller of the system. It polls the robot and human positions every 10 *ms* using the pins *poll* and *read*. From these inputs, the distance between human and robot is computed and the correct operation mode is selected. Altogether, there are three operation modes that are defined as



(a) ESM of *Control Unit*. (b) ESM of *Robot Operation*.

Figure 3: ESMs of building blocks *Control Unit* and *Robot Operation*

follows:

- *Normal mode:* If no human is closer than 25 meters to the robot, the robot accelerates with $5\frac{m}{s^2}$ until reaching a speed of $10\frac{m}{s}$. When it is 11 m close to its endpoint, it decelerates with $5\frac{m}{s^2}$ until reaching a speed of $1\frac{m}{s}$ which is carried until reaching the endpoint.
- *Yellow mode:* If a human is detected in a distance of less than 25 meters but more than 10 meters, the robot is slowed down with a rate of $10\frac{m}{s^2}$ until reaching a speed of $2\frac{m}{s}$ (resp. $1\frac{m}{s}$ if it is closer than 11 m to its endpoint).
- *Red mode:* If the human is within 10 meters range to the robot, the robot makes an emergency stop with a deceleration of $15\frac{m}{s^2}$.

The behavior of the building block *Control Unit* is specified by its ESM depicted in Fig. 3(a). An ESM is a UML state machine describing which of its parameter nodes are passed by tokens in a certain transition. *Control Unit* is initiated by parallel flows through the parameter nodes *startRobot* and *startPerson* which contain the initial positions of robot and human. Thereafter, the building block is in state *active* in which the environment (symbol / right of the parameter node identifier) may send position data via parameter nodes *senseRobot* and *sensePerson* while from the block itself (/ left of the parameter node identifier) the current operation mode may be sent via *setNormal*, *setYellow* and *setRed*. The building block is terminated and all remaining tokens on its activity are removed if the activity containing the call behavior action of the block is terminated as well which is described by the transition /.

Figure 4 shows the UML activity of building block *Robot Operation*. It contains the building block *Mode Selector* storing the current operation mode. *Robot Controller* models the controller of the robot, which chooses the current robot speed according to the operation mode and the position of the robot in

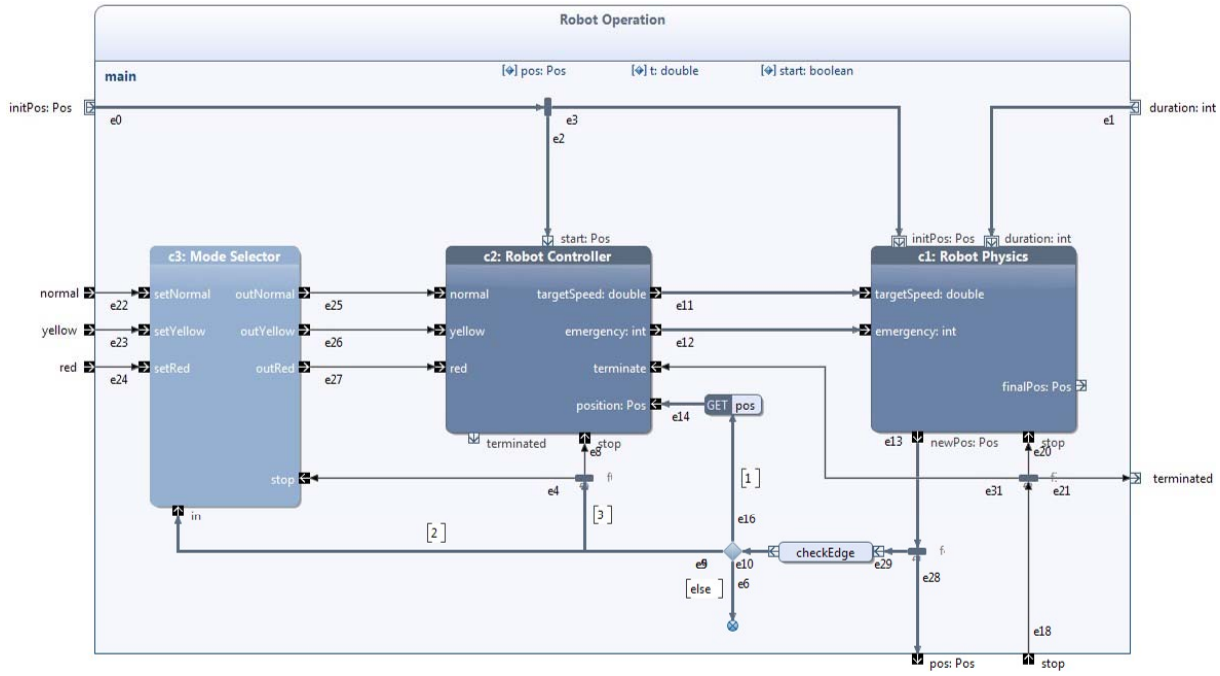


Figure 4: UML activity of building block *Robot Operation*

the factory hall. Using the block *Robot Physics* we model the simulator for the robot. The continuous behavior is specified by a difference equation which is executed every 5 *ms*.

The ESM of block *Robot Operation* in Figure 3(b) determines that the block is started by parallel token flows via the parameter nodes *duration* and *initPos* which refer to the execution time of the difference equation (i.e., 5 *ms*) and the initial position of the robot. In state *active*, the operation modes *normal*, *yellow* or *red* may be received by the environment while by *pos* the current position of the robot may be forwarded towards the sensor in block *Control Unit*. The block is terminated by a token coming via parameter node *stop* which leads to an output via *stopped*. This signal leads to the termination of all inner blocks followed by the termination of *Robot Operation*.

The third block of the example is called *Suicidal Human*. This somehow odd name refers to the simulation of a human attempting to approach the robot as fast as humanly possible. So, it describes the worst case situation to be solved by the safety controller. Like the robot, the behavior of the human is specified by a difference equation that is executed every 5 *ms*. For the sake of brevity, we do neither list this block nor the other blocks of our system in detail here.

After creating and composing all building blocks of our system, we can check them for the presence of functional design errors like not fulfilling their ESMs (see [17]). Further, if all checks are passed, Reactive Blocks automatically generates executable Java of our system which can be carried out to simulate robot runs.

2.1 Probability Assumptions and Temporal Safety

As discussed above, the factory hall, in which humans and machines collaborate in close proximity, is monitored by camera sensors for collision avoidence. A safety controller constantly monitors the

Table 1: Accumulative probability distribution of the execution times for the different tasks

Delay Type	**Maximum Time**	**Probability**	**Fig. 5**	**Fig. 6**
Time to fetch sensor data including polling delay	15 *ms*	10 %		
	17 *ms*	40 %		
	18 *ms*	85 %		
	19 *ms*	99.998 %		
	20 *ms*	100 %		
Processing time recognition unit	250 *ms*	10 %		
	260 *ms*	30 %		
	270 *ms*	60 %		
	280 *ms*	90 %		
	285 *ms*	99 %		
	290 *ms*	100 %		
Communication time recognition unit to robot	15 *ms*	80 %	r1	
	16 *ms*	98 %	r2	
	16.5 *ms*	99.5 %	r3	
	16.9 *ms*	99.99999995 %	r4	
	20 *ms*	100 %	r5	
Internal robot processing time and actuator reaction	150 *ms*	5 %		r1
	159 *ms*	90 %		r2
	160 *ms*	95 %		r3
	165 *ms*	99.9995 %		r4
	170 *ms*	100 %		r5

operation for the proximity of humans and then decides which operation mode to choose. Of course, to avoid collisions when the robot is still moving, we have to guarantee maximum reaction times for the different functions carried out in order to slow down or stop the robot. The four main sub-tasks are the fetching of sensor data including the delay between two pollings of the sensors, the processing time of the safety controller in order to compute the distances between human and robot and to decide about the correct operation mode, the communication delay between the safety and the robot controller as well as the processing time of the robot controller including delays within the robot starting to break.

For elaborating our approach, we assume that the probabilities for the reaction times associated with the sub-tasks correspond to the percentages depicted in Table 1. They show the probability that a task is finished within a certain time in an accumulative way. For instance, according to the table, the fetching of the sensor data is finished within 15 *ms* with a probability of 10 % while the overall probability that it is completed within 17 *ms* is 40 %. Thus, we guarantee that this task is carried out within 20 *ms*. Of course, in practice one cannot give axiomatic guarantees of real-time properties since a control system is always subject to external influences like a failure of the computer hardware running it. We decided to ignore these kinds of external error in our models but are aware that, when our tool chain is used for real hazard analysis, such faults have to be taken into account as well. The values in the table do not correspond to an existing system, but rather represent typical values one might expect in some field-bus based systems.

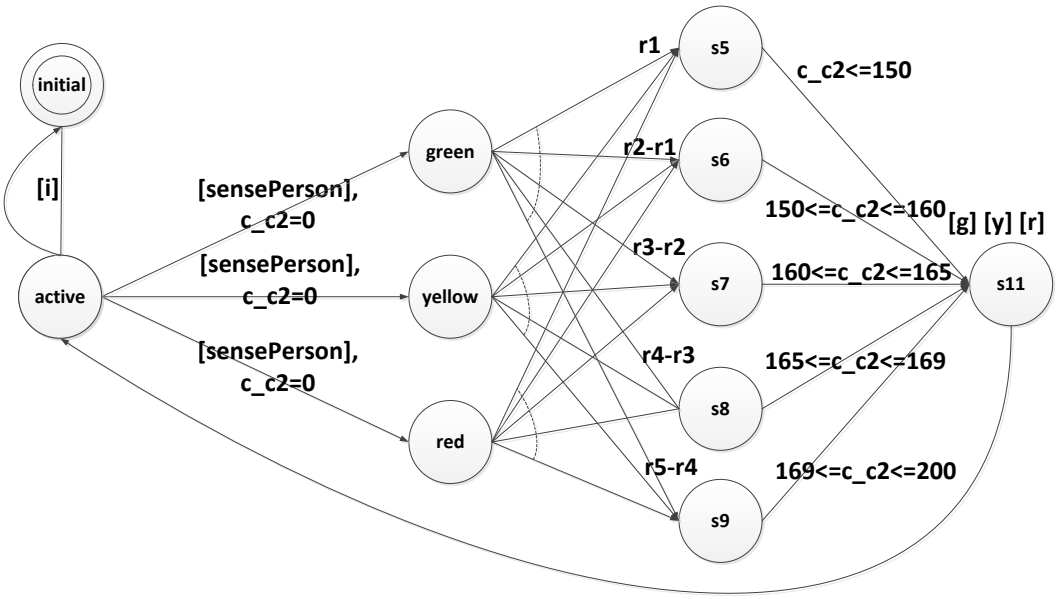


Figure 5: PRTESM for the *ControlUnit* block

3 Probabilistic Real-Time Extended State Machines

Following the concept of Timed Automata [1], we extended our external state machines (ESM) to Real-Time ESMs (RTESM) in [12, 13]. RTESMs allow the specification of deadlines for the time a building block may rest in a certain RTESM state. As a new contribution in this paper, we introduce the further extension of the RTESMs to Probabilistic Real-Time External State Machines (PRTESM). Extending the habitual pattern in Reactive Blocks to model functional and non-functional interface properties of building blocks, the PRTESMs make the description of probabilistic real-time behavior possible and allow to describe discrete probability distributions like the ones listed in Table 1. PRTESMs allow a straightforward transformation into Probabilistic Timed Automata (PTA) [22] that can be model checked by verification tools like PRISM [19, 20].

Figures 5 and 6 show the PRTESMs of the blocks *Control Unit* and *Robot Operation*. To facilitate the transformation into PTAs, a PRTESM contains an initial state *initial* representing both the initial and final states of the corresponding ESM. Moreover, the PRTESM may contain special states that express probabilistic behavior of the concerned actions as well as the synchronization semaphores and timed constraints used to model real-time properties. In the PRTESMs listed in Figures 5 and 6, the values **r1**, **r2**, **r3**, **r4**, **r5** represent the probabilities from the third resp. forth section in Table 1. The time deadlines are measured in 100 microseconds.

The approach for the generation of a PTA from a PRTESM for a real-time blocks is semi-automatic.

- First, a set of communication actions are identified in the building block concerning the underneath hardware or communication protocol. In our example in the building block *Control Unit*, the parameter pins *setNormal, setYellow, setRed* and *sensePerson* realize the communication among distributed agents in the moving robot scenario. The *setX* set of parameters realize communication between robot controller and robot actuator while the *sensePerson* parameter realizes the communication between camera sensor and robot controller. Thus we extract the probabilistic real-time

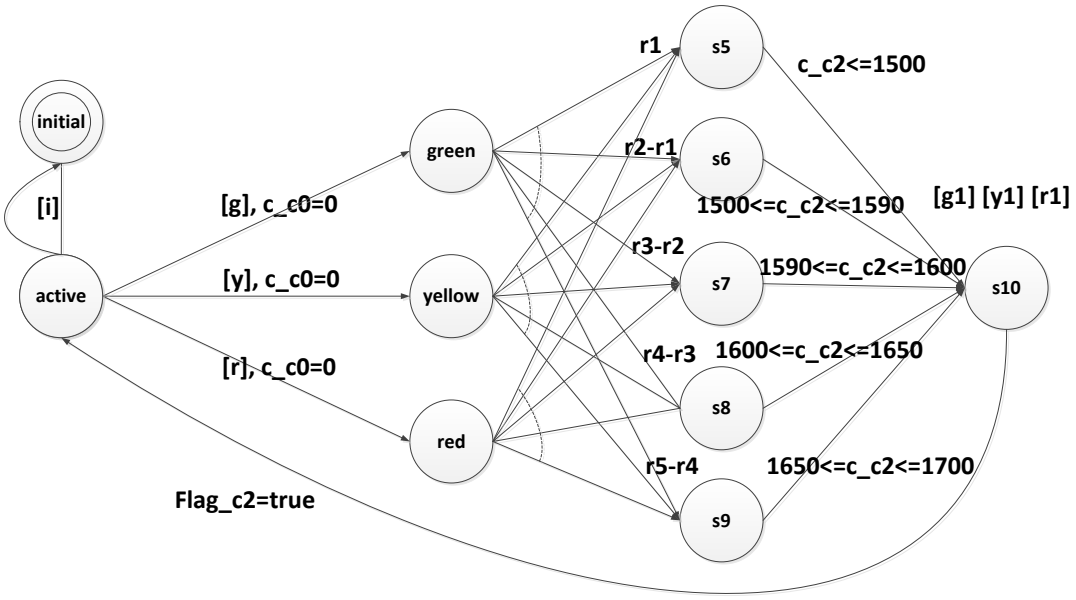


Figure 6: PRTESM for *RobotOperation* block

actions in the system to PRTESM and ignore other actions.

- Second, transitions corresponding to distributed communications are transformed to new states and transitions expressing probabilities. In our example these are: $active \rightarrow_{setNormal} active$, $active \rightarrow_{setYellow} active$, $active \rightarrow_{setRed} active$ and $active \rightarrow_{sensePerson} active$.

The code excerpt in Figure 7 corresponds to the PRTESM in Figure 5 and illustrates the corresponding Probabilistic Timed Automata (PTA) of the building block *Control Unit*. The formalism declaration *pta* demands that the following modules follow the timed automata style. Time and probability values are declared as constant values before the module declaration. *c2_Control_Unit_prtesm* is the module name. A PTA transition in PRISM is started with a pair of brackets (*[]*) and optional synchronization commands in the brackets, e.g., a semaphore *i* initializes distributed building blocks simultaneously during the system initialization, and semaphores *r,y* are abbreviated from communication parameter *setRed, setYellow* to synchronize module *Control Unit* (robot controller) and module *Robot Operation* (robot actuator). ESM transitions which are labeled as real-time probabilistic actions are exported and extended with probabilistic description. Line *14* to *17* gives an example command in PRISM showing the probabilities. It declares that when *s_c2* variable equals to 2 it has *80%* possibility of going to state 5, *98%-80%* possibility of going to state 6. When state 5 is reached, guard conditions demand that clock *c_c2* must be no greater than *150* (representing the system delay in 100 microseconds) and not smaller than *160* (see also Table 1).

4 Probabilistic Spatial Property Verification

Probabilities in system models can affect the spatial behavior of systems. Depending on the specification — as provided by Reactive Blocks — we can determine areas in time and space which a system component is likely to occupy or interact with.

```
1. pta
2.  const int c2_1 = 150; // time unit 0.0001 s
3.  const int c2_2 = 160;
4.  ...
5. const double r1 = 0.8;  // probability
6. const double r2 = 0.98; // accumulative probability
7. module c2_Control_Unit_prtesm
8.         s_c2 : [0..10] init 0;
9.         c_c2 : clock;
10.        flag_c2 : bool init false;
11.        [i] s_c2=0 -> (s_c2'=1);
12.        [r] s_c2=1 -> (s_c2'=2)&(c_c2'=0);
13.        ...
14.        [sensePerson] s_c2=1 -> (s_c2'=2)&(c_c2'=0);
15.        [] s_c2=2 -> r1 : (s_c2'=5) + r2-r1 : (s_c2'=6)
16.        + r3-r2 : (s_c2'=7) + r4-r3 : (s_c2'=8)
17.        + r5-r4 : (s_c2'=9);
18.        [y] s_c2=5&c_c2<=c2_1 -> (s_c2'=10);
19.        [y] s_c2=6&c_c2>=c2_1&c_c2<=c2_2 -> (s_c2'=10);
19.        ...
20.        [] s_c2=10 -> (s_c2'=10) & (flag_c2'=true);
21.endmodule
```

Figure 7: Excerpt of PTA codes corresponds to Figure 5.

In our robot system the distribution of latencies for reacting to the detection of a human can result in different areas indicating the possible positions of a robot for given time points, each one associated with a probability. Another example is varying speed. If the speed sensors of the robot are not accurate, they may come with a distribution of a possible error. Therefore the robot may accelerate to a speed slightly higher or lower than the specified 10 m/s. As depicted in Fig. 8, this inaccuracy leads to a wider area, the robot may be in at a certain point of time, and the sizes of the areas increase over the distance the robot moves. Furthermore, one can relate the area sizes also with probabilities. Following the discrete probability distribution in Table 1, in Fig. 9 we show the varying areas covered by the robot. With a probability of 80 % it will be within the orange rectangle at a selected point of time, with 90 % in the dark yellow one and with certainty in the light yellow one.

For collision analysis we may neglect probabilities that are below a certain threshold defining residual risks that one is willing to bear. In our example, we can prove that there is indeed a situation that a person running into the factory hall with a speed of $10\frac{m}{s}$, may hit the robot before it completely stopped. According to the distribution in Table 1, the risk for this, however, is not higher than $5 \cdot 10^{-14}$. Since we found out by simulating the situation that the speed of the robot at such an impact is $0.625\frac{m}{s}$ at most, the collision risk is extremely low and the impact essentially not different from the human running into a stationary object.

We implemented BeSpaceD, a tool [6] for checking spatial behavior of spatiotemporal systems as well as an input language for such systems. The implementation is done in Scala and comprises abstract datatypes that indicate spatial availability, interaction or occupation in areas in a coordinate system for time intervals or timepoints. It is possible to give parameterized specifications describing nondeterministic systems and their spatial behavior.

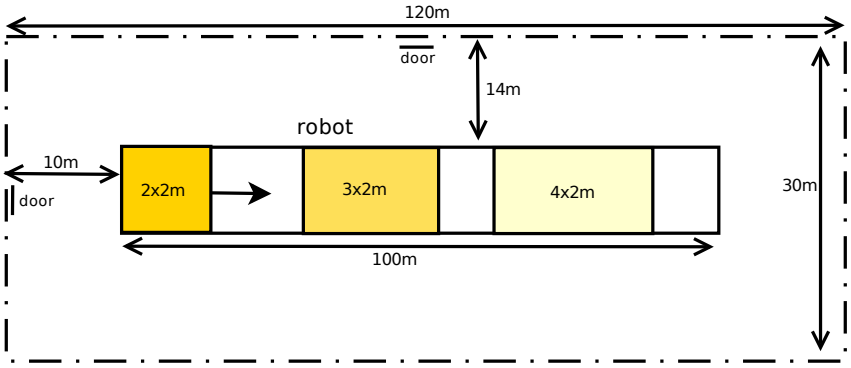


Figure 8: Possible space occupation induced by unknown speed

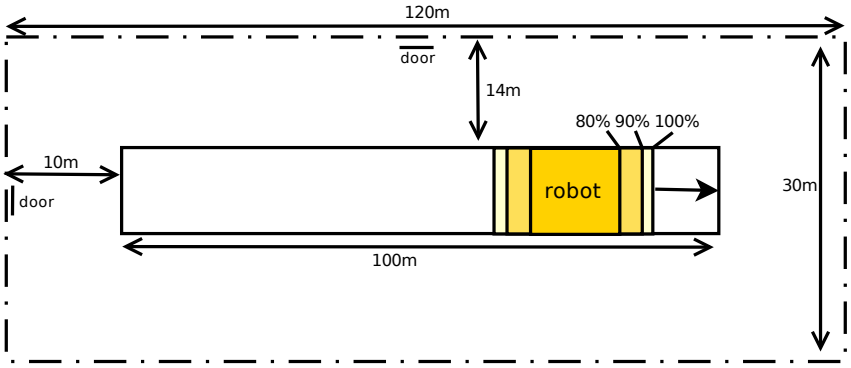


Figure 9: Space occupation and probability

For this work, however, we restrict ourselves to the checking of scenarios generated from simulation runs of Reactive Blocks models. Particularly, in a simulator run we stored every five milliseconds a tuple consisting of the current time stamp and the positions of human and robot in a format readable by BeSpaceD. Thereafter, we use BeSpaceD to detect collisions and other spatial interactions for the various scenarios and probabilities. In this way, we found out the situation mentioned above that a human indeed may collide with the robot before it has stopped. We are able to learn such space-related safety issues already while modeling the system. This makes it much easier to adapt the system functionality or to impose stricter real-time properties if non-bearable situations were detected.

The specification of each spatial entity in a scenario has the form:

$time = t \longrightarrow$
occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'
$time = t + 1 \longrightarrow$
occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'

....

$time = t + n \longrightarrow$
occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'

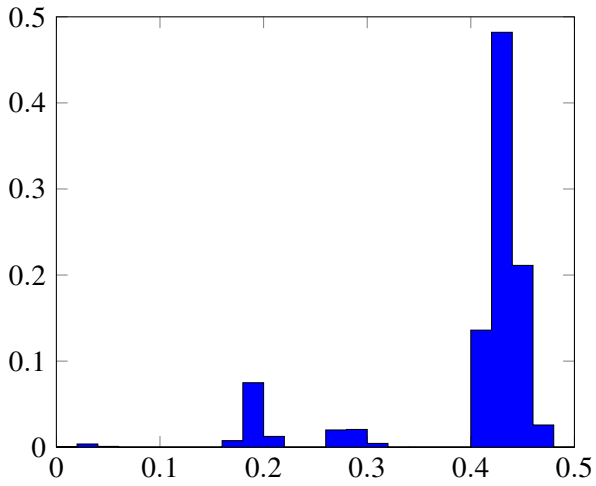


Figure 10: Probability density function for the system

Probabilities are treated as attributes to an occupied spatial area. Different means to check spatial properties — here collisions — formulated over these inputs are provided in BeSpaceD and are based on SAT and SMT solving and direct Scala implementations.

5 Probability Distributions as Verification Results

We verify probabilistic properties based on the probabilistic timed temporal logic PTCTL [25]. The probability operator $P_{=?}$ allows reasoning about numerical probabilistic values, and is supported by the so called stochastic games engine [21] in the verification tool PRISM. The results of our property verification are probability distributions. Figure 10 shows the non-accumulative probability distribution of the overall system reaction time displayed in a histogram. The x axis represents time values with a frequency of 0.02 s. The y axis shows the numerical probability value.

In our analysis, we verified a set of probabilities expressed in PTCTL as follows:

$$P_{=?}[F_{\leq T}\ ''target''](T \in [0.0,...0.5])$$

In the above temporal specification formula, the operator F is a path operator that equals to *eventually* in LTL and can be used inside the P operator. The pattern $F_{\leq T}$ stands for "*within T time unit*". The logical expression *"target"* inside the PTA models that the parallel composed real-time probabilistic actions are indeed executed. The formula expresses the possibility that within T time units, the labeled actions are achieved.

Important for us are questions like: Is the robot reaction time no more than a designed safe time limit of 0.46 s? Checking that a reaction time of 0.46 s is enough to avoid a collision is discovered by simulation using Reactive Blocks and BeSpaceD. With PTCTL we check this property using the following formula:

$$P_{=?}[F_{\leq 4600}\ ''target'']$$

The result indicates that when a human enters the monitored area, the robot reacts within 0.46 s with the probability of 99.99874114988752 %. Due to the potential severity of a collision, this number does

not seem sufficient. We discussed in Sect. 4 that the impact will not be serious with a reaction time of 0.5 s. Of course, if the robot controller reacts within 0.47 s, the maximum speed at impact will be even lower (i.e., $0.125\frac{m}{s}$ at most according to our simulations) and the risk of injuries is seen to be remote. Considering this fact, we assume that the layout of our example system as sufficiently safe.

6 Related Work

Here, we present related work regarding formal methods, safety analysis, probabilities and spatial verification.

In [14] an algorithm is presented to check whether a given Markov Chain satisfies formulas of Probabilistic Computation Tree Logic (PCTL). PCTL and related verification techniques are typically applied to analyze the reliability of timed systems. The work presented in [20] presents an algorithm for the verification of probabilistic real-time systems annotated with discrete probability distributions. This can be seen as a starting point for our work.

Safety analysis is a critical phase in the development of the systems we are aiming at. In [24], a solution for incorporating safety requirements in software architecture is provided. Means facilitating Model-Driven Architecture based development and safety analysis are presented. The work presented in [11] aims at providing a modeling framework for the hazard analysis of component-based systems. The authors intended to include traditional hazard analysis techniques, e.g., Fault Tree Analysis (FTA), but found its inappropriateness for component based systems. Thus, new techniques like State Event Fault Tree (SEFT) are proposed. The new proposed model is suitable for describing stochastic behaviors with the help of existing tools such as Matlab/Simulink. These can be extended to support such models. In [23] an extension of concurrent Kleene algebras is provided. An axiomatisation for probabilistic, concurrent and nondeterministic systems is presented, and the simulation and verification of probabilistic automata can be carried out. In the past we have studied the application of probabilistic models in a theorem prover for guaranteeing fault-tolerance of embedded systems [3].

A process algebra-like formalism for describing and reasoning about spatial behavior has been introduced in [7, 8]. Process algebras come with a precise formal semantics and target the specification of highly parallel systems. Another logic-based approach to describe spatial areas is the Region Connection Calculus (RCC) [2]. It includes spatial predicates to describe the separation and connection of areas. The area of hybrid systems has features the development of different tools for reasoning and verification. SpaceEx [10] allows the modeling of continuos 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. Additionally, it is possible to model spatial behavior in more general purpose oriented verification tools in Hybrid systems (e.g., [26]).

7 Conclusion

In this paper, we proposed an approach for the model-driven development of probabilistic real-time systems with highly reusable compositional building blocks that incorporate discrete probability distributions for describing stochastic time behaviors. These extensions are used to predict system performance as well as probabilistic safety properties. In addition, we established a development tool set both for temporal and spatial probabilistic behaviors of systems. The Probabilistic Real-Time External State Machines (PRTESM) of building blocks give a straightforward view of stochastic real-time behaviors of component-based systems. Software architects and safety engineers can use this for verification and

analysis. The limitations of the approach is that some intral or inter components' communications are, instead, described as messages passed and received in UML, such that such behaviors can not be captured by ESMs. Also, whether this approach is applicable depends on the abstraction level. In the future, we want to further study and emphasize the compositionality of probabilistic spatial behavior definitions, i.e., of systems composed of appropriately logically detailed subsystems. We plan to advance the description logic as well as the introduction of a probabilistic spatial behavioral type infrastructure that extends previous work [4, 5].

Acknowledgments We like to express our thanks to Song Zheng Song from the National University of Singapore (NUS) for the useful discussion during the research work.

References

[1] R. Alur, D. Dill. Automata for Modeling Real-Time Systems, in Automata, Languages and Programming, pages 322–335, vol. 443 of LNCS, Springer-Verlag, 1990, doi:10.1007/BFb0032042.

[2] B. Bennett, A. G. Cohn, F. Wolter, M. Zakharyaschev. Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. Applied Intelligence, 17(3), Kluwer Academic Publishers, November 2002, doi:10.1023/A:1020083231504.

[3] J. O. Blech. Probabilistic Compositional Reasoning for Guaranteeing Fault Tolerance Properties. 15th International Conference On Principles Of Distributed Systems, Toulouse, France, vol. 7109 of LNCS, Springer, December 2011, doi:10.1007/978-3-642-25873-2_16.

[4] J. O. Blech. Towards a Framework for Behavioral Specifications of OSGi Components. Formal Engineering approaches to Software Components and Architectures. Electronic Proceedings in Theoretical Computer Science, 2013, doi:10.4204/EPTCS.108.6.

[5] J. O. Blech, Y. Falcone, H. Rueß, B. Schätz. Behavioral Specification based Runtime Monitors for OSGi Services. Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), vol. 7609 of LNCS, Springer-Verlag, 2012, doi:10.1007/978-3-642-34026-0_30.

[6] J. O. Blech and H. Schmidt. Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems. Improving Systems and Software Engineering Conference, Melbourne, Sep. 2013.

[7] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). Information and Computation, 186(2), Nov. 2003, doi:10.1016/S 0890-5401(03)00137-8.

[8] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). Theoretical Computer Science, 322(3), pp. 517–565, Sep. 2004, doi:10.1016/j.tcs.2003.10.041.

[9] L.A. Gunawan, P. Herrmann. Compositional Verification of Application-Level Security Properties. Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS 2013), pages 75-90, Paris, vol. 7781 of LNCS, Springer, Feb/Mar 2013, doi:doi:10.1007/978-3-642-36563-8_6.

[10] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler. SpaceEx: Scalable Verification of Hybrid Systems. Computer Aided Verification (CAV'11), 2011, doi:doi:10.1007/978-3-642-22110-1_30.

[11] L. Grunske, B. Kaiser and Y. Papadopoulos. Model-driven safety evaluation with state-event-based component failure annotations. In CBSE, pages 33–48, vol 3489 of LNCS, Springer, 2005, doi:10.1007/11424529_3.

[12] F. Han, P. Herrmann, and H. Le. Modeling and verifying real-time properties of reactive systems.' 18th International Conference on Engineering of Complex Computer Systems (ICECCS), 2013, doi:10.1109/ICECCS.2013.13.

[13] F. Han. P. Herrmann. Modeling real-time system performance with respect to scheduling analysis. Proceedings of the 6th IEEE International Conference on Ubi-Media Computing, Nov 2013.

[14] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing, 6(5). 1994, doi:10.1007/BF01211866.

[15] D. Henriques, J.G. Martins, P. Zuliani, A. Platzer, E.M. Clarke. Statistical Model Checking for Markov Decision Processes. Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on , vol., no., pp.84,93, 17-20 Sept. 2012, doi:10.1109/QEST.2012.19.

[16] F.A. Kraemer, P. Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009), pages 571-585, Denver, vol. 5795 of LNCS, Springer-Verlag, Oct. 2009, doi:10.1007/978-3-642-04425-0_44.

[17] F. A. Kraemer, V. Slåtten, and P. Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. Journal of Systems and Software. vol. 82, no. 12, pp. 2068–2080, December 2009, doi:10.1016/j.jss.2009.06.057.

[18] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07), Eds. M. Bernardo and J. Hillston, pp. 220–27, vol. 4486 of LNCS (Tutorial Volume), Springer, 2007, doi:10.1007/978-3-540-72522-0_6.

[19] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. Proc. 23rd International Conference on Computer Aided Verification (CAV'11), pp. 585–59, vol. 6806 of LNCS, Springer, 2011, doi:10.1007/978-3-642-22110-1_47.

[20] M. Kwiatkowska, G. Norman, R. Segala and J. Sproston. Automatic Verification of Real-time Systems with Discrete Probability Distributions. Theoretical Computer Science, 282, pages 101-150. June 2002, doi:10.1007/3-540-48778-6_5.

[21] M. Kwiatkowska, G. Norman and D. Parker. Stochastic Games for Verification of Probabilistic Timed Automata. Technical report RR-09-05, Oxford University Computing Laboratory. June 2009, doi:10.1007/978-3-642-04368-0_17.

[22] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant Systems (FORMATS/FTRTFT'04), Eds. Y. Lakhnech and S. Yovine, pp. 293–308, vol. 3253 of LNCS, Springer, 2004, doi:10.1016/j.ic.2007.01.004.

[23] A. McIver, T. Rabehaja, and G. Struth. Probabilistic concurrent kleene algebra. Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, Rome, 23rd-24th March 2013, Eds. L. Bortolussi and H. Wiklicky, pp. 97–115, vol. 117 of Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2013, doi:10.4204/EPTCS.117.7.

[24] M.A.de Miguel, J. F. Briones, J. P. Silva, A, Alonso. Integration of safety analysis in model-driven software development. Software, pp.260-280, vol.2, no.3, , IET, June 2008, doi:10.1049/iet-sen:20070050.

[25] D. Parker. Verification of Probabilistic Real-time Systems. Proc. 2013 Real-time Systems Summer School (ETR'13). August 2013.

[26] A. Platzer, J-D. Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). International Joint Conference on Automated Reasoning, pp. 171–178, LNCS 5195, Springer, 2008, doi:10.1007/978-3-540-71070-7_15.

[27] V. Slåtten, F. A. Kraemer, and P. Herrmann. Towards Automatic Generation of Formal Specifications to Validate and Verify Reliable Distributed Systems: A Method Exemplified by an Industrial Case Study. Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering (GPCE11), pp. 147–156, ACM, 2011, doi:10.1145/2047862.2047888.