Modeling Spatial Aspects of Safety-Critical Systems with FocusST

Maria Spichkova[1], Jan Olaf Blech[1], Peter Herrmann[2], and Heinz Schmidt[1]

[1] RMIT University, Melbourne, Australia
{maria.spichkova, janolaf.blech, heinz.schmidt}@rmit.edu.au
[2] Norwegian University of Science and Technology (NTNU), Trondheim, Norway
herrmann@item.ntnu.no

Abstract. This paper presents an approach for modeling and verification of components controlling behaviour of safety-critical systems in their physical environment. In particular, we introduce the modeling language FocusST that is centred on specifying time and space aspects. Verifications can be carried out using the interactive semi-automatic proof assistant Isabelle. The approach is exemplified by means of a railway system scenario.

1 Introduction

Many safety-critical systems (SCSs) consist of mobile units autonomously moving in their physical environment. Modeling such systems requires not only the definition of the software part but also a specification of interactions with the physical environment. In consequence, the models need to capture timing and spatial aspects that should provide a basis for formal verification of safety properties. In most cases, however, we do not need the whole representation of an SCS but only those parts relevant to a concrete purpose. Thus, an appropriate model should give an overview of core system properties and allow an effective inconsistencies funding, reducing modeling and verification effort.

For modeling SCSs suitably, it is essential to have a well developed theory covering real-time and space requirements since mistreating or excluding them can lead to specification errors due to difficulties of choosing a correct abstraction. Moreover, in many cases reasoning about time to represent a real-time system makes the specification more readable (in comparison to an untimed representation), simplifies the argumentation about its properties, and gives a formal basis for verification. A suitable representation of SCSs should also make it possible to model information flow not only in time but also in space, because the spatial aspect may influence the delays of interactions between subcomponents of the system as well as between the system and the environment. This point is important for cost reduction of interoperability testing at the integration phase of the development process. Further, for a versatile application of a notation, the selection of a suitable space-time coordinate system should be relatively free.

The modeling language that we use in our approach is FocusST. It allows us to create concise but easily understandable specifications and is appropriate

for application of the specification and proof methodology presented in [19, 25]. This methodology allows writing specifications in a way that carrying out proofs is quite simple and scalable to practical problems. In particular, a specification of an SCS can be translated to a Higher-Order Logic and verified by the interactive semi-automatic theorem prover Isabelle [17] also applying its component Sledgehammer [4]. Sledgehammer employs resolution based first-order automatic theorem provers (ATPs) and satisfiability modulo theories (SMT) solvers to discharge goals arising in interactive proofs. Another advantage is a well-developed theory of composition as well as the representation of processes within a system [20]. The collection of FOCUSST operators over timing aspects and their properties specified and verified using the theorem prover Isabelle is presented in the Archive of Formal Proofs [21]. In this work we focus on modeling of spatial aspects.

Related Work: One of the most well-established models for the specification and verification of real-time system design is timed automata, introduced by Alur and Dill [1, 2]. A timed automaton is a finite automaton extended by real valued clocks that are applied to measure the time elapsed since certain events occurred. The clocks are used in so-called clock invariants that restrict the time, a timed automaton may rest in a particular state without executing certain transitions. Timed automata assume perfect continuity of clocks which may not suit the purposes of the work presented here, especially if we deal with an embedded system with instantaneous reaction times. Furthermore, they do not prevent Zeno runs [12], i.e., executing an infinite number of transitions in a finite period of time. To solve this, the idea of robust model checking was introduced by Puri [18] and revised in other approaches, e.g., [8]. In this paper, we suggest another solution: We use asynchronous channels between timed automata and argue about possibly infinite *message sequences* towards an automaton at some time interval. This can be represented by using an infinite sequence of finite time intervals as input for a timed automaton. Any timed transition system can be discretised without loss of generality [14]. For this reason, we apply a discrete model of time where any granularity defining the concrete meaning of a time interval according to the system requirements can be used. We can even switch from one time granularity to another using predefined operators. A great advantage of the proceeding is that it excludes Zeno runs.

Related work regarding spatial aspects has been done with respect to logic and tools. A process algebra like formalism for describing and reasoning about spatial behavior has been introduced in [10, 11]. Process algebras come with a clear and formal semantics definition and are aimed towards the specification of highly parallel systems. Here, disjoint logical spaces are represented in terms of expressions by bracketing structures and carry or exchange concurrent processes. Results on spatial interpretations can be found in [15]. Many aspects of spatial logic are in general undecidable. A quantifier-free rational fragment of ambient logic (corresponding to regular language constraints), however, has been shown to be decidable in [26]. Work on spatial model checking by ourselves is presented in [6, 7]. Furthermore, this approach was coupled with the model-based

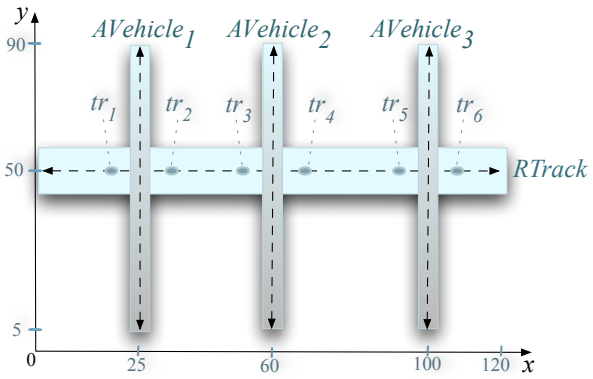

Fig. 1. Automatic transport system

engineering technique Reactive Blocks [16] such that reactive systems (e.g., SCS controllers) can be developed using models and be checked for spatial properties before generating executable code [13].

Scenario: Due to the well specified degrees of freedom enforced by rail tracks, trains have been a popular target for verification work (see, e.g., [3]). This makes them also an appropriate target to introduce the main idea of modeling and verification of spatial aspects. Here, we present a small example from this area that is depicted in Fig. 1. A train *RTrack* shuttles on a rail track that is crossed by three roads. On each road, an autonomous vehicle *AVehicle* is operating. All four mobile units are characterized by a number of constraints on their location, speed, movement direction, etc. To avoid collisions, the train sends a *wait* signal to the respective *AVehicle* while passing one of the corresponding critical points tr_i close to the crossings ($1 \leq i \leq 6$). If a critical point is ahead of a crossing in the direction of the train, the *wait* signal expresses a time interval, for which the *AVehicle* has to stop if it is heading towards the crossing and is far enough to stop in due time. The time interval may depend on the speed of the train giving it sufficient time to pass the crossing. If a critical point is behind a crossing, *wait* contains value 0 indicating that the vehicle may immediately continue moving. The fact that an *AVehicle* is only stopped for a certain time interval at most, provides a challenge on the space-related behavior since we have to guarantee that the train already left the crossing when the time interval passed. This poses the following questions: How should we model spatial properties of this system in a readable way? Does this modeling technique allow formal verification of safety properties? Is this model also appropriate to specify and verify a large number of components, e.g., for the case that we have not three but one thousand *AVehicle* components? In this paper we will answer these questions by presenting our modeling approach in FOCUSST.

2 Spatial Aspects in FocusST

The FocusST language was inspired by Focus [9], a framework for formal specification and development of interactive systems. In both languages, specifications are based on the notion of *streams*. However, in the original Focus input and output streams of a component are mappings of natural numbers $\mathbb{N}$ to single messages,whereas a FocusST stream is a mapping from $\mathbb{N}$ to lists of messages within the corresponding time intervals. Moreover, the syntax of FocusST is particularly devoted to specify spatial (S) and timing (T) aspects in a comprehensible fashion, which is the reason to extend the name of the language by ST. The FocusST specification layout also differs from the original one: it is based on human factor analysis within formal methods [22, 23].

We specify every component using assumption-guarantee-structured templates. This helps avoiding the omission of unnecessary assumptions about the system's environment since a specified component is required to fulfil the guarantee only if its environment behaves in accordance with the assumption. In a component model, one often has transitions with local variables that are not changed. Also, outputs are often not produced, e.g., when a component gets no input or some preconditions necessary to produce a nonempty output are violated. In many formal languages this kind of invariability has to be defined explicitly in order to avoid underspecified component specifications. To make our formal language better understandable for programmers, we use in FocusST so-called *implicit else-case* constructs. That means, if a variable is not listed in the guarantee part of a transition, it implicitly keeps its current value. An output stream not mentioned in a transition will be empty. Further, we do not require to introduce auxiliary variables explicitly: The data type of a not introduced variable is universally quantified in the specification such that it can be used with any data value.

The FocusST specifications are a special form of timed automata that we name *Timed State Transition Diagrams* (TSTDs). A TSTD can be described in both diagram and textual form. For easier argumentation, we can further represent it by a special kind of tables including a number of new operators that work on time intervals. For a real-time system S with a syntactic interface $(I_S \triangleright O_S)$, where I_S and O_S are sets of timed input and output streams respectively, a TSTD corresponds to a tuple $(State, state_0, I_S, O_S, \rightarrow)$, in which $State$ is a set of states, $state_0 \in State$ is the initial state, and $\rightarrow \subseteq (State \times I_S \times State \times O_S)$ represents the transition function of the TSTD.

An input action for a TSTD is the set of current time intervals of the input streams of the system, while the output action is the set of corresponding time intervals of the output streams of the system. Focus distinguishes between *weak causal systems* and *strong causal systems* (see [9]). In the former case, the output must be produced within the same time interval the input is consumed while in the latter one the output has to be produced within a delay of at least one time unit. The exact delay needs to be defined according to the timing requirements on the specified system.

Spatial Aspects: In addition to the representation of timing properties in the language, we define a special type of components specifying real objects that can physically change their location in space, so-called *sp-objects*. Each sp-object is associated with three special variables storing its current *location* (i.e., central point of the object), *speed* and *direction* of movement. For simplicity, the variable *speed* is defined over the set of natural numbers $\mathbb{N}$, while *location* is of type *Space* and defines a coordinate having two or tree dimensions according to the system's needs. In our two-dimensional example, *Space* is a tuple of two Cartesian coordinates *xx* and *yy*. Finally, *direction* is defined over the type *Directions* $= \{0, \ldots, 359\}$ which represents the angle in the Cartesian coordinate system. In comparison to the local variables declared within components, these three variables are global and can be used to specify physical interaction of components in a system.

A system model may be constrained by restricting the directions and speed of an sp-object. This allows us to verify whether the specified behaviour excludes the possibility that the object enters restricted areas during time intervals marked as dangerous, e.g., collisions with other sp-objects.

FOCUSST *specification:* Figure 2 depicts the textual representation of the sp-object specification pattern for the component *AVehicle* introduced in the scenario section of the introduction. This component is strong causal with a delay of one time unit, and has the three input channels *wait* and *tSpeed* of type $\mathbb{N}$ as well as *tDir* of type *Directions* declared in the interface part of the specification using label "in". The ports *tSpeed* and *tDir* are used to notify changes of the target speed and the target direction of the object. If *AVehicle* is too close to a potential obstacle, e.g., the crossing with *RTrack*, it is signalled via the *wait* port to stop for a number of time units. Thereafter it continues moving with the previous speed. Label "out" defines the output channel *resp* of type *Event* that consists a single element *event* used to signal the start of motion by the vehicle.

Let us name some of the operators used to specify time intervals in our streams: $\langle\rangle$ denotes an empty list, i.e., a single time interval without any events, and $\langle x \rangle$ a list consisting of the element x; $\mathsf{ft}.l$ describes the first element of a list l; s^i represents the ith time interval of the stream s.

Empty brackets after the component's name mean that *AVehicle* does not have any parameters. The component uses the local variable *timer* referring to the current timer value (the special value 0 means that the timer is not active while 1 indicates that it has to time out). By *lspeed*, we store the speed the object carried before needing to stop. The variable *timer* is initially set to be 0 while the initial value of *lspeed* is not specified.

The keyword "asm" lists the assumption, *AVehicle* demands from its environment, i.e., specified using the FOCUSST operator msg_1, at most one message is received via each of the ports *wait*, *tSpeed* and *tDir* at any time interval.

The section "gar" contains the transitions and other formulas. Here, variable settings before executing (i.e. at some time interval t) a transition are marked by simple variable identifiers, e.g., *timer*, while the operator $'$ refers to their setting afterwards, e.g., $timer'$ denotes the value of the timer variable at the time

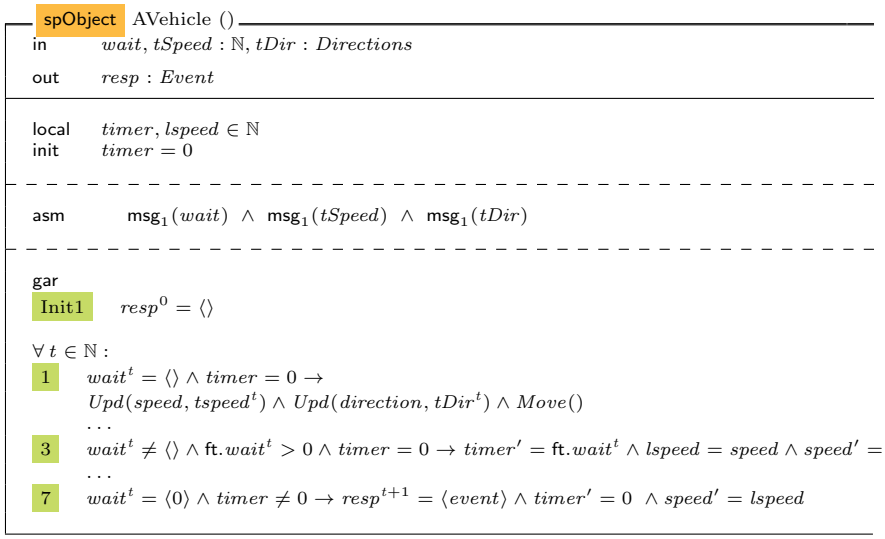

Fig. 2. Textual representation of specification pattern *AVehicle*

interval $t+1$. The initial condition Init1 specifies that the output channel *resp* is empty at the beginning. Formula 1 models a transition that if *AVehicle* does not receive a wait-signal at the time interval t and its timer is inactive, the component moves according its target direction and speed values. This is expressed by the function $Move()$ that updates the value of the variable *location*. The function *Upd* defines the updates of the variables *speed* and *direction* according to the events in the input streams *tSpeed* and *tDir* at time interval t. Moreover, due to the implicit else-case construct discussed above, the timer does not change its value and the output stream *resp* is an empty list. Formula 3 specifies the start of the timer if *RTrack* approaches the crossing. In this case, *AVehicle* receives a number $k > 0$ via the port *wait*. The timer is started by setting its variable to k while the vehicle is stopped ($speed' = 0$) and the previous speed is stored in the auxiliary variable *lspeed*. Transition 7 specifies that if *AVehicle* receives 0 via the port *wait* (i.e., it does not need to wait for the *RTrack* any more) while its timer is active, the timer will be set off and, at the next time interval, the *event*-signal will be sent indicating that the vehicle resumes moving again into its target direction. We omit here the rest of the specification due to lack of space.

Fig. 3 shows the diagrammatic version of the TSTD for *AVehicle*. It contains all transitions that we label with the same numbers as in the textual representation. To increase readability of the graphical representation, we distinguish three types of the transition labels by coloured representation: Inputs and constraints on the current local variables' values are marked blue, outputs and changes of

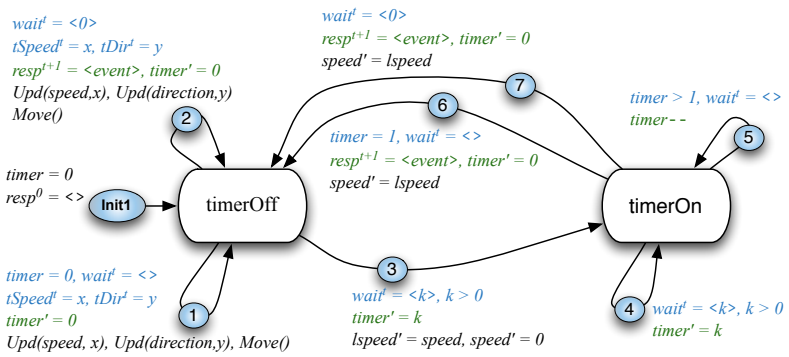

Fig. 3. Timed State Transition Diagram for *AVehicle*

general (non-spatial) local variables' values green, and changes of spacial aspects black.

3 Verification Constraints on Spatial Aspects

We can restrict directions and speed of an sp-object by adding constraints. Predicates are associated with every component. Per default, they are specified as true but can be restricted to represent precise bounds of a component. To calculate whether a collision of sp-objects is possible, we assign to each sp-object a global constant *rad* that describes radius of the maximal space the object can "cover" in the worst case. The maximal space the object can occupy at the time t, is denoted by the variable *rzone* of the type *Zone*. This variable is defined as a tuple $(minX, minY, maxX, maxY)$ of natural numbers which are calculated according to the values of *speed*, *location* and *rad*. The *rad* value of a composite component is defined by analysing which space its subcomponents can occupy in the worst case: $S.rad = max(WCX, WCY)/2$ with WCX and WCY being the maximum extensions of all of the subcomponents of S in direction x resp. y.

We represent the set of all the components' constraints by a table, to increase readability and to check schematically whether constraints on a *composed component* correspond to the constraints on its subcomponents, e.g.,

$$\forall\, S,\, C : C \in subcomp(S) \rightarrow$$
$$(S.rzone.minX \leq S.C.rzone.minX \wedge S.rzone.minY \leq S.C.rzone.minY) \wedge$$
$$(S.rzone.maxX \geq S.C.rzone.maxX \wedge S.rzone.maxY \geq S.C.rzone.maxY)$$

$$\forall\, k,\, S,\, C : C \in subcomp(S) \rightarrow$$
$$(k \leq S.rzone.minX \rightarrow (k + S.C.rad) \leq S.C.location.xx)$$

To analyse spatial properties, we need to specify the rules how the locations of the objects can change over time. The location of the sp-object C at the

Table 1. Set of the spatial constraints

Component	rad	$locationRestr$	$speedRestr$	$directionRestr$
$AVehicle_1$	2	$location.xx = 25$	$speed < 10$	$direction = 90 \vee direction = 270$
$AVehicle_2$	2	$location.xx = 60$	$speed < 15$	$direction = 90 \vee direction = 270$
$AVehicle_3$	2	$location.xx = 100$	$speed < 20$	$direction = 90 \vee direction = 270$
$RTrack$	4	$location.yy = 50$		$direction = 0 \vee direction = 180$

beginning of the next time interval can be computed from its speed, direction of movement and the current location. In particular, we specify a *trajectory* of the object during a time interval t to be a straight line, thus, it can be described by the coordinates of two locations, at the beginning of the current and the next time interval: $C.tr = [C.location, C.location']$. Then, we can describe the space where the object C can be *during* the time interval t (let denote it $C.rzoneInterval$) by a set of coordinates:

$$\{(a, b) \mid \exists (a_1, b_1) \in C.tr \wedge a_1 - C.rad \leq a \leq a_1 + C.rad \wedge b_1 - C.rad \leq b \leq b_1 + C.rad\}$$

If $C.speed = 0$ holds, the component C does not move (i.e., $C.location' = C.location$) such that $C.rzoneInterval$ describes the *rzone* space in this case.

Restrictions on location, speed and direction can be specified both pointwise and by using minimum and maximum limits, where a variable can have any value within the defined interval. Let us explain this by using the scenario introduced in Sect. 1. Its spatial contraints are listed in Tab. 1. It is easy to see from this table and Fig. 1 that the four mobile units can occupy 100 units on the x coordinate and 85 units on the y coordinate. Thus, $S.rad$ is assigned with the value 50. By defining an additional constraint for the space that can be used by the overall example system S, we implicitly restrict the corresponding constraints of its components, e.g.,

$$0 \leq S.rzone.minX \wedge S.rzone.maxX \leq 120 \wedge$$
$$5 \leq S.rzone.minY \wedge S.rzone.maxY \leq 90$$

implies that for the component $AVehicle_1$ holds not only $location.xx = 25$ but also $7 \leq location.yy \leq 88$ (taking its own value $rad = 2$ into account). Representing the spatial aspect of a component as a pair of coordinates and a radius, we specify possible collisions between two objects C_1 and C_2 during the time interval t by $PCollision^t(C_1, C_2)$. Thus, an important property for this system is that collisions between the *RTrack* and *AVehicle* components are excluded, i.e. for all $t \in \mathbb{N}$ and $i, j \in \{1, 2, 3\}$ the following holds

$$\neg PCollision^t(RTrack, AVehicle_i) \wedge \neg PCollision^t(AVehicle_i, AVehicle_j)$$

Since in our example, the *AVehicle* objects move on parallel roads, the property on the right side holds trivially. If a *possible* collision is detected, the corresponding case should be analysed carefully both on an abstract (logical) and

on a physical level: Due to our overapproximation of space, not every situation labelled as possibly dangerous on the abstract level indeed corresponds to a real physical collision but, on the other side, any real physical collision must be detectable on the abstract level.

We have used the interactive theorem prover Isabelle/HOL [17] to analyse whether collisions between the *RTrack* and *AVehicle* components are excluded. For example, direction and location constraints together with behaviour specifications imply that the mobile units can collide, if we underspecify the coordinates of the critical points $tr_1, \ldots, tr_6$ as well as the initial locations.

Due to similarity of specifications of separate components, the model of the presented system is scalable not only for specification but also for verification purposes. Even if we have not three but thousand *AVehicle* components, proofs of their spatial behavioural properties can be reused or generated.

4 Conclusions

This paper presents the FOCUSST approach for modeling and verification of safety-critical systems using specifications based on time intervals and spatial aspects. Several features have been demonstrated using an example system based on interacting autonomous vehicles. We focus on timing and spatial aspects as well as readability of the specifications and ease of verification of core properties. For the proofs, we have applied the interactive semi-automatic proof assistant Isabelle.

Our future research direction comprises work on the modeling levels for SCSs, that reflect the idea of remote integration/interoperability testing in a virtual environment [5, 24] as well as automatisation of proof generation for spatial behavioural properties from the system model. Moreover, we want to combine this verification technique with the modeled-based engineering tool Reactive Blocks [16] to facilitate the practical development of the control software for space-aware SCSs.

References

1. R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. R. Alur and P. Madhusudan. Decision Problems for Timed Automata: A Survey. In *SFM*, pp. 1–24, 2004.
3. P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor: A Successful Application of B in a Large Project. *Formal Methods (FM'99)*, vol. 1708 of *LNCS*, Springer, 1999.
4. J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT Solvers. *Journal of Automated Reasoning* 51(1):109–128, 2013
5. J. O. Blech, M. Spichkova, I. Peake, H. Schmidt. Cyber-Virtual Systems: Simulation, Validation & Visualization. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2014)*, 2014.

6. J. O. Blech and H. Schmidt. BeSpaceD: Towards a Tool Framework and Methodology for the Specification and Verification of Spatial Behavior of Distributed Software Component Systems. In *arXiv.org*, `http://arxiv.org/abs/1404.3537`, 2014.
7. J. O. Blech and H. Schmidt. Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems. In *Improving Systems and Software Engineering Conference*, 2013.
8. P. Bouyer, N. Markey, and O. Sankur. Robust Model-checking of Timed Automata via Pumping in Channel Machines. *Formal Modeling and Analysis of Timed Systems*, Springer, 2011.
9. M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.
10. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186(2), 2003.
11. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). *Theoretical Computer Science*, 322(3):517-565, 2004.
12. R. Gómez and H. Bowman. Efficient Detection of Zeno Runs in Timed Automata. *Formal Modeling and Analysis of Timed Systems*, Springer, 2007.
13. F. Han, J. O. Blech, P. Herrmann, and H. Schmidt. Towards Verifying Safety Properties of Real-Time Probability Systems. In *Formal Engineering approaches to Software Components and Architectures (FESCA)*, EPTCS, 2014.
14. T. Henzinger, Z. Manna, and A. Pnueli. What Good are Digital Clocks? In *Colloq. on Automata, Languages and Programming*, pp. 545–558. Springer, 1992.
15. D. Hirschkoff, É. Lozes, D. Sangiorgi. Minimality Results for the Spatial Logics. In *Foundations of Software Technology and Theoretical Computer Science*, LNCS 2914, Springer, 2003.
16. 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*, 82(12):2068–2080, 2009.
17. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS 2283, Springer, 2002.
18. A. Puri. Dynamical Properties of Timed Automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.
19. M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, TU München, 2007.
20. M. Spichkova. Focus on Processes. Tech. Report TUM-I1115, TU München, 2011.
21. M. Spichkova. Stream Processing Components: Isabelle/HOL Formalisation and Case Studies. In *Archive of Formal Proofs*, ISSN 2150-914x, 2013.
22. M. Spichkova. Human Factors of Formal Methods. In *IADIS Interfaces and Human Computer Interaction (IHCI)*. 2012.
23. M. Spichkova. Design of Formal Languages and Interfaces: "Formal" Does Not Mean "Unreadable". *Emerging Research and Trends in Interactivity and the Human-Computer Interface*. IGI Global, 2013.
24. M. Spichkova, H. Schmidt, and I. Peake. From Abstract Modelling to Remote Cyber-Physical Integration/Interoperability Testing. In *Improving Systems and Software Engineering Conference*. 2013.
25. M. Spichkova, X. Zhu, and D. Mou. Do We Really Need to Write Documentation for a System? In *International Conference on Model-Driven Engineering and Software Development*, 2013.
26. S. Dal Zilio, D. Lugiez, and C. Meyssonnier. A Logic You Can Count on. In *Symposium on Principles of programming languages*, ACM, 2004.