Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems

Jan Olaf Blech and Heinz Schmidt
RMIT University
Melbourne, Australia

Abstract

In this paper, we present work and ideas towards a framework for modeling and checking behavior of spatially distributed component systems. Components can communicate and interact with each other and show external spatial and communication behavior. For example, a car driving on a road and communicating with other cars can be modeled as a component. Our framework aims at checking of behavioral properties like collision detection using verification tools. We present examples related to cyber-physical systems as a motivation for our work.

1 Introduction

Different techniques have been introduced for modeling and checking of cyber-physical systems and their properties. These comprise various means for modeling cyber-physical systems like differential equations, automata and different notions of time like continuos or discrete time. Based on these modeling approaches tools for checking properties have been developed.

On the other hand, the software engineering community has been studying the design and architecture of component based systems for decades. Specification formalisms like automata and message sequence charts are frequently used to describe expected behavior of systems. Different means for checking these specifications are available.

In this work, we motivate work towards a unified approach to cyber-physical systems and component based software development. We motivate a new framework that aims at a seamless specification process for spatial cyber-physical behavior and communication and interaction behavior between different components. We present work towards a framework that is especially suited for large scale widely distributed systems like cars driving on a road network. Several components may be distributed over a large space and have distinct features of communication, spatial and internal behavioral aspects. The second part of this work proposes a process for checking properties of our models. A goal is to enable the handling and checking of properties of large models in a highly parallel fashion. A medium term goal is the possibility to process our models in a cloud based environment and offer services around them.

We study modeling and verification scenarios which are characterized by a discrete notion of time that features timestamps. Timestamps are partially ordered. This allows for the modelling of synchronous and asynchronous systems with distinct synchronisation points between different components. Technically our approach is aimed at handling models and checking their properties in a highly parallel environment based on cloud-infrastructure.

Overview We present related work in Section 2. Our motivating and guiding examples are presented in Section 3. The methodology for modeling our systems is motivated in Section 4 and the principles of checking properties are described in Section 5. A discussion on future work and a conclusion is featured in Section 6.

2 Related Work

Work that is relevant to this paper has been done in areas like formal methods, hybrid-systems, software engineering and robotics.

A process algebra like formalism for describing and reasoning about spatial behavior has been introduced in [8] and [9]. 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. For results on spatial interpretations see, e.g., [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 [29].

Special modal logics for spatial-temporal reasoning go back to the seventies. The Region Connection Calculus (RCC) [3] includes spatial predicates of separation. For example RCC features predicates indicating that regions do not share points at all, points on the boundary of regions are shared, internal contact where one region is included and touches on the boundary of another from inside, proper overlap of regions, and proper inclusion. In addition [3] 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 [21] allows the modeling of continuos hybrid systems based on hybrid automata. It can be used for computing overapproximations of the space occupied by a moving – in time and space – object. Additionally, it is possible to model spatial behavior in more general purpose oriented verification tools in Hybrid systems (e.g., [24]).

The notion of time and component interaction used in this paper is compatible with the petri nets induced notion. This is used, e.g., in the BIP framework [2] for modeling of distributed asynchronous systems. Invariants are used as an intermediate step for verification in the BIP context [4]. Invariants are also a key intermediate representation of components in this work.

Related to our work is the work on path planning for robots (e.g., [17, 18]). In our work, however, we are concentrating on checking existing properties of systems rather than optimization or discovery of new possible paths. Collision detection for robots in combination with motion planning has been studied for a long time, see, e.g., [16] and [10]. Strongly related to motion planning is the task of efficiently handling geometric reasoning. On this geometric interpretation level, techniques have been investigated to structure the tasks of detecting possible inference between geometric objects (e.g., [14] and [19]) for efficient analysis.

Data models for cyberphysical infrastructure in construction, plant automation and transport – domains that we are aiming for in this paper – have been studied in the past. Unlike in this paper, many of the existing real-world applications are aligned towards a geometric representation of componentes and are typically based on so-called 2.5 dimensional GIS representations where the 3rd dimension $z = f(x, y)$ is represented as a function f of the 2 dimension x and y coordinates. This [1] limits the geometric, topological and information retrieval use of such models. True three dimensional modeling is far from common practice [26]. Our approach is not limited to a particular geomteric representation, coordinate or dimension system. Future extensions of interest include consideration of standards such as the Web 3D Services and the Sensor Web Enablement Architecture of the Open Geospatial Consortium [1], visualisation and decision support [28] and efficient data structures for fast meta reasoning and presenting subproblems of choice to specialist solvers used in our examples.

We have been investigating mathematical models of behavior in space in previous work [27], which also features a survey on work related to these models. Specification and reasoning on existing software component systems [6, 5] has also been examined by us. In this work we aim towards a unified view of these aspects and the introduction of a behavioral type style mechanism to annotate components which is a future goal of this work. A software architectural model was proposed in [25]. Symbolically reasoning about invariants of asynchronous distributed systems, which is a part of our verification methodology in this paper, has been studied by us in [7].

3 Motivating Examples

We present two motivating examples to show the scenarios we are targeting and introduce some basic principles. When formalising the behavior of system components, different aspects like occupied space, physical interactions, data communication, and internal states can be distinguished. These aspects can change depending on time and interaction with other components. Properties of a particular aspect do not necessarily have to depend on other aspects. For example, spatial behavior may be independent of data communication aspects. To exemplify this, we present two

[1] `http://www.opengeospatial.org`

different models.

3.1 Concurrent Window Cleaning

Figure 1 shows a concurrent window cleaning system. Platforms d_1, d_2 are attached to a mobile device on a roof. They can be moved horizontally by moving the mobile device on the roof and vertically through a rope. Attached to each platform is a robot arm: t_1 and t_2. It can have different positions relativ to its platform, but only within a limited range. Furthermore, we can have some kind of internal state for each platform.

The behavior of each robot is controlled by a program. This program implies behavior which is potentially non-deterministic and may depend on other robots or external events. The behavior is characterized by spatial aspects, the movement of the robot, communication aspects (e.g., interactions with other robots or some external controlling device), and internal state changes.

We are interested in simulating possible window cleaning scenarios. Robots have local states and when they interact with another robot their actions may undergo a synchronization. This synchronization does not need to be global, i.e., in case of many robots cleaning a window, the synchronisation does not have to be shared with all robots. For this reason, we only have a partial order of time. Each element of this partial order is called a *timestamp*.

One aspect that we are interested in, is whether a certain system state implies a collision. In order to do this, we could examine the position of each platform and each robot arm. Based on this, we could calculate the exact space boundaries that each device uses. An alternative way is to use an abstraction and use an overapproximation of the space used. More coarse grained abstractions may only have to take the position of the platforms into account.

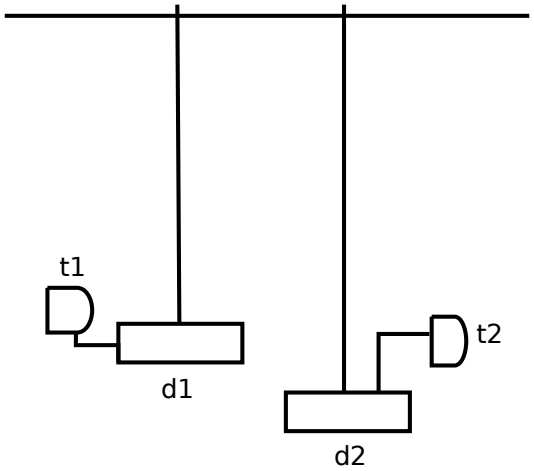

Figure 1: Concurrent Window Cleaning

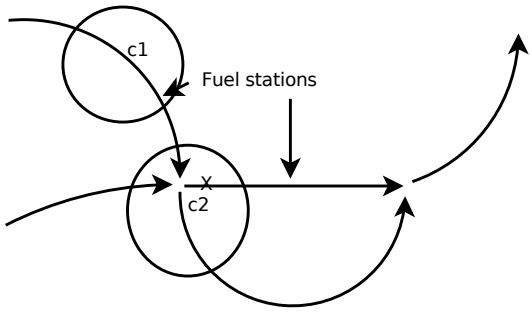

Figure 2: Communicating cars

3.2 Communicating Cars

Figure 2 shows two cars driving on a network of roads. The road network is formalized as a graph. Cars have a limited ability of locally communicating with each other, iff they are within a certain distance of each other. This distance is indicated by a circle around the car in the figure. A car has a local state. This is defined by the following ingredients:

- The road section it is currently driving on: the edge of the corresponding graph.
- The position on the road, e.g., indicating the progress on the road section and the direction it is facing.
- Aspects of the internal state that encapsulates the parts of a state that are truly local to the car like the amount of fuel left.
- Communication aspects of an internal state, e.g., states that are reached throughout an ongoing communication effort with other cars according to a protocol.

One can see, that spatial, truly internal and communication aspects are encoded in the states.

For calculating the distance between two cars travelling on different roads, we need an interpretation. We take the road edges and the position on the road into account. The topological graph information needs to be interpreted in a geometrical way. For example, we can retrieve data from a geographic information system to get the exact location on earth that allows us to com-

pute the distance between the two cars. Alternatively a car may itself keep track of the actual coordinates within its state. Furthermore, we need a shared timestamps between the two cars so that we have access to their positions at the same time.

The local communication possibilities of the cars may be used for various purposes. For example, a car might inform the another car that a road has been blocked (indicated by the X in the figure). The road blockage may leave a fuel station inaccessible. The other car may therefore alter a scheduled fuel stops and use another fuel station.

4 Modeling Methodology

This section describes the key elements of modeling systems and representing desired properties in our framework. We propose the use of four different layers for modeling systems and their behavior in terms of space, internal behavior and communication.

1. A topological or geospatial coordinate system. In our window cleaning example this is provided by the window itself. In the communicating cars example this is provided by the topological graph representing the roads and its geographic / geometric interpretation.

2. Static components and their interconnections. In the sense of for example a street network, we can have road blocks and road construction sites as well as refuel stations. In the window cleaning example, we can have single obstacles on the surface.

3. Mobile components. These comprise the cars and robots from our examples moving within or across their allocated segment without changes in the static components structure.

4. Information flow. This can occur between mobile and static components such as cars or robots or a fuel station.

Systems can evolve and change their structure and behavior. A new road may be added or a new fuel station. Cars can move and communicate with each others. The layers are ordered with respect to the rate of expected changes, with layer 1 seeing the least changes and level 4 most changes over time. Mobile components and static components can be structured into subcomponents.

4.1 Modeling of the Topological Layer

We propose to model the topological layer as a graph (N, E) comprising a set of nodes N and edges E between these nodes. N and E can be infinite. A geometrical interpretation may be defined ontop of this graph, e.g., by introducing a function measuring the distance between two nodes: $N \times N \to \mathbb{R}^+$.

In the case of the window cleaning example the graph is made up from imaginary points distributed over the window plane, e.g., realizing the intersections of a raster. The edges are possible transition path between them.

In case of the communicating cars, the topological layer reflects the road structure. However, the graph from the topological layer is much larger than the graph imposed by the road structure, because different positions on roads will become independent nodes.

4.2 Modeling of Static Components

Static components are described by their position on the topological layer. Multiple nodes in the topological layer may be occupied. A static component can feature an automaton (L, l_0, C, E, T) comprising a set of locations L, an initial location $l_0 \in L$, a set of conditions C, a set of events E and a set of transitions $(l, c, e, l') \in T$ representing a transition between $l, l' \in L$, guarded by a condition $c \in C$ and annotated with a set of events $e \subseteq E$.

The automaton may be used to describe a communication protocol updates to the internal state or minor changes in the occupation of space. Interactions are realized using guards and events.

4.3 Modeling of Mobile Components

Like static components, mobile components can comprise an automaton (L, l_0, C, E, T). The automaton may be realized to describe a communication protocol. In addition to this C and E can comprise conditions and events related to conditions in the topological layer and movements inside this layer. Compared to static components, mobile components will see much more updates to their spatial behavior over time. In terms of tool based verification and modeling it can be desirable to treat them independent from static components.

4.4 Modeling of Information Flow

Information flow may be encapsulated in the guards and events from the components. Conditions on the information flow, like the fact that communication is only possible within a certain range can be realized using guards, too.

4.5 Modeling Properties

Properties have to be formally encoded so that verification tools can handle them. In addition to this, our framework shall support the formulation of human readable properties at a higher specification level. Here, we list three different levels of properties that can be regarded.

- On the most basic level, we have *Logical formula*. Formally, they can be written as LTL or CTL formula, regular expressions or simply first or higher order logic formula. These formula are build from *atomic expressions* that represent events encountered by components, conditions on internal states, and positions taken by components in space. In case of LTL, CTL and regular expressions constraints on timestamp orders are encoded in the operators of the logic. In contrast to this, first or higher order logic formula can contain explicit information on timestamps which are atomic expressions in this case, too. Atomic expressions are connected using the logical operators of the underlying logic to form larger expressions. For reuse purposes, these can be tailored to one of the specification layers or distinct entities within these layers.

- On the next level, we are interested in properties that can be formulated in a textual way, e.g., in a requirements document like the detection of possible *collisions* between mobile objects and whether mobile objects are close enough in order to interact or *communicate* with each other. These properties can be formalized using one or multiple formula from the basic level.

- Eventually these properties can be combined to ask questions like: can a task be safely executed within a certain amount of time or is it advisable to refuel before the next road intersection, because a road may be blocked?

Translating high-level properties to low level formula may be supported by domain-specific knowledge and transformation templates gained from experience.

5 Checking Methodology

Checking properties of models is to be supported by a tool based infrastructure in our framework. We describe the basic notions of invariants and verification conditions and transformations on them that are central to our checking methodology. We introduce the tool based workflow and present examples.

5.1 Invariants

Invariants are abstractions of the behavior of components or distinct aspects of a system. They capture an overapproximation of a component's behavior or a system aspect with respect to spatial, communication and internal behavior over time. Logically invariants are represented as first or higher-order logic formula that contain predicates representing atomic events, conditions on space and communication and timestamps.

Since invariants provide an overapproximation of a component they may serve as an basis for checking of safety properties. Checking results achieved on the basis of our invariants can be carried back to the original system if some preconditions are met (see e.g., [20]). This means that we can check system properties on the invariant level and be sure that the results hold for the original system.

Handling invariants instead of complete specifications can be much easier, since details that are hard to check or are even undecidable can be abstracted.

5.2 Verification Conditions

While invariants are logical formula that aim to capture the semantics of a component or system aspect, verification conditions are logical formula that are aimed as input for a distinct verification tool or algorithm. They must obey to the distinct format of the tool, e.g., a SAT formula in case of a SAT solver [2].

5.3 Invariant and Verification Condition Transformation

In our methodology, we need to transform invariants, merge or split invariants and generate verification con-

[2] e.g., by using Sat4j: `http://www.sat4j.org/`

ditions from invariants. Invariants and verification conditions are build from the elements of the underlying logic or specification language and thus come with a term structure. Transformations, which are defined inductively on the term structure of invariants and verification conditions are part of our methodology. Transformations can be realized as functional programs or – especially if additional features like higher-order unification and the verification of the transformation itself is an issue - in a higher-order theorem prover like Coq [3] or Isabelle/HOL [23].

5.4 The Workflow

Figure 3 shows the workflow for checking the properties of models with respect to involved computation steps and tools. Models and properties are given to our tool chain for checking.

- In a first step, an algorithm is used to identify large scale verification goals. For example static components that are separated in space and do not interact with each other may be checked independently. The behavior of mobile components may also be checked independently if they are only acting in a local area in space or time. Properties may only regard a local area and therefore only the behavior of components that act within this local area has to be taken into account. Best practices for this approach can be scenario and domain specific. For determining whether one component or a property depends on another, techniques from model checking like cone-of-influence reduction [12] can be applied. A result of this step is the identification of a set of relevant components for the desired properties.

- In a next step, we compute the invariants for all relevant components. One way to do this is to unfold all possible execution traces of an automaton, include all relevant events and annotate them with timestamps. These traces are than formalized as invariants.

- We use these invariants to generate verification conditions in a next step. Verification conditions can be checked by separate highly specialised tools like SMT solvers (e.g., Yices [13] or z3 [22]).

[3] `http://coq.inria.fr/`

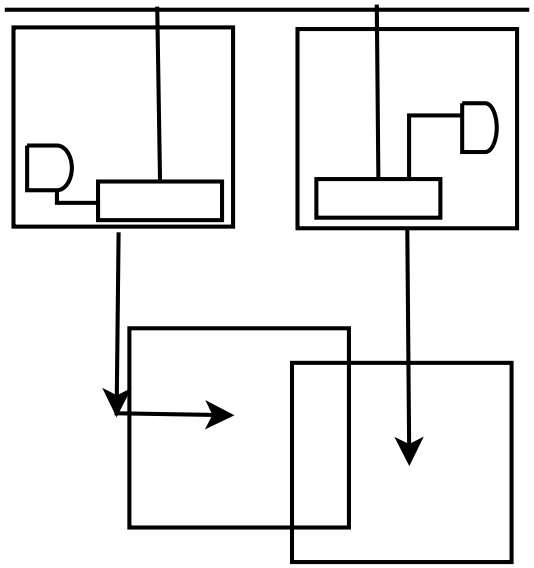

Figure 4: Concurrent Window Cleaning Invariant Collision

- The last step collects the results of the specialised tools and presents an overall results. Optionally, we refine invariants or verification conditions, if the result does not satisfy our needs. Eventually this may lead to an iterative process like a fixpoint computation or counterexample guided abstraction refinement [11] where invariants are the predicates.

Computation of invariants can be done in parallel in many cases. Verification conditions never depend on each other, so they can be always checked in parallel.

5.5 Window Cleaning Example: Invariants

Here, we present steps necessary for checking the window cleaning example. Time in the the window cleaning example is synchronous.

Figure 4 shows a collision between boxes that represent invariants of the two window cleaning platforms. The invariants themself are parameterized by time and represent overapproximation of the platforms' spatial behaviors. Each box represents a distinct timestamp (some intermediate timestamps are omitted in the figure). Our goal is to formalize the behavior of the platforms, so that we can automatically detect the collision.

A textual representation for an example invariant for the first window cleaning platform can be seen in Figure 5. An invariant for the second window platform is shown in Figure 6. The general structure of invariants

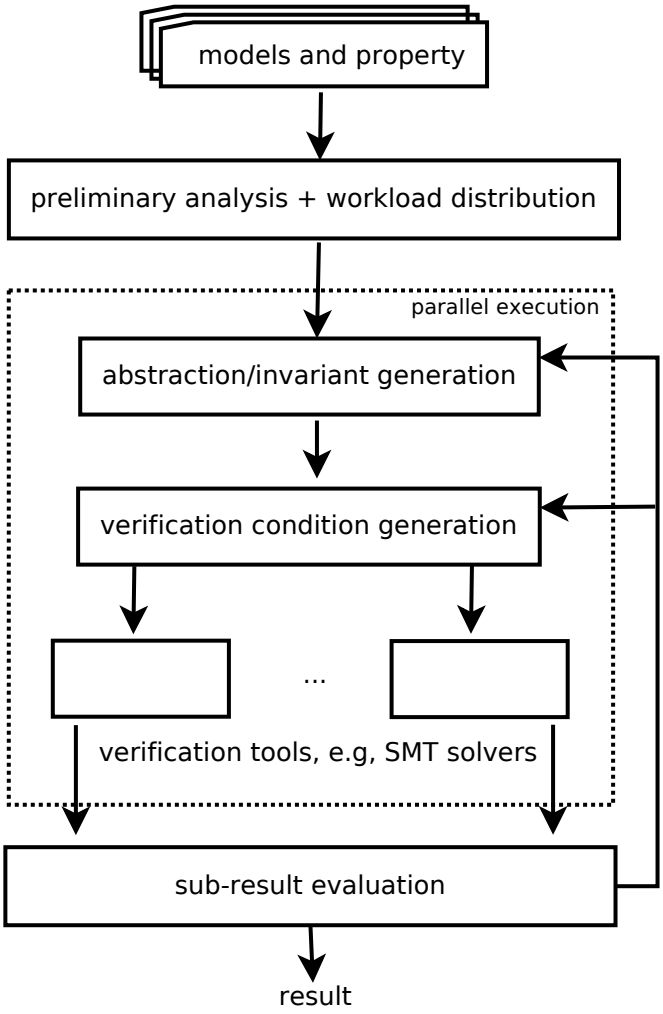

Figure 3: Tool workflow

$$
\begin{aligned}
&I_{c_1}(t) = \\
&(t=0) \rightarrow occupyXYspace(87.0, 0.0, 97.0, 10.0) \wedge \\
&(t=1) \rightarrow occupyXYspace(87.0, 1.1, 97.0, 11.1) \wedge \\
&(t=2) \rightarrow occupyXYspace(87.0, 2.3, 97.0, 12.3) \wedge \\
&(t=3) \rightarrow occupyXYspace(87.0, 3.8, 97.0, 13.8) \wedge \\
&(t=4) \rightarrow occupyXYspace(87.0, 5.2, 97.0, 15.2) \wedge \\
&(t=5) \rightarrow occupyXYspace(87.0, 6.0, 97.0, 16.0) \wedge \\
&(t=6) \rightarrow occupyXYspace(87.0, 7.0, 97.0, 17.0) \wedge \\
&(t=7) \rightarrow occupyXYspace(87.0, 8.7, 97.0, 18.7) \wedge \\
&(t=8) \rightarrow occupyXYspace(87.0, 9.9, 97.0, 19.9) \wedge \\
&(t=9) \rightarrow occupyXYspace(87.0, 10.9, 97.0, 20.9) \wedge \\
&(t=10) \rightarrow occupyXYspace(90.0, 12.0, 100.0, 22.0) \wedge \\
&(t=11) \rightarrow occupyXYspace(91.1, 12.0, 101.1, 22.0) \wedge \\
&(t=12) \rightarrow occupyXYspace(92.0, 12.0, 102.0, 22.0)
\end{aligned}
$$

Figure 5: Simplified example invariant (first platform)

$$
\begin{aligned}
&I_{c_2}(t) = \\
&(t=0) \rightarrow occupyXYspace(100.0, 0.0, 110.0, 10.0) \wedge \\
&(t=1) \rightarrow occupyXYspace(100.0, 1.2, 110.0, 11.2) \wedge \\
&(t=2) \rightarrow occupyXYspace(100.0, 2.5, 110.0, 12.5) \wedge \\
&(t=3) \rightarrow occupyXYspace(100.0, 3.8, 110.0, 13.8) \wedge \\
&(t=4) \rightarrow occupyXYspace(100.0, 5.2, 110.0, 15.2) \wedge \\
&(t=5) \rightarrow occupyXYspace(100.0, 6.3, 110.0, 16.3) \wedge \\
&(t=6) \rightarrow occupyXYspace(100.0, 7.9, 110.0, 17.9) \wedge \\
&(t=7) \rightarrow occupyXYspace(100.0, 9.1, 110.0, 19.2) \wedge \\
&(t=8) \rightarrow occupyXYspace(100.0, 10.0, 110.0, 20.0) \wedge \\
&(t=9) \rightarrow occupyXYspace(100.0, 11.2, 110.0, 21.2) \wedge \\
&(t=10) \rightarrow occupyXYspace(100.0, 12.6, 110.0, 22.6) \wedge \\
&(t=11) \rightarrow occupyXYspace(100, 13.9, 110, 23.9) \wedge \\
&(t=12) \rightarrow occupyXYspace(100, 15, 110, 25)
\end{aligned}
$$

Figure 6: Simplified example invariant (second platform)

can be seen. On a top level, we make a case distinction on timestamps t. Here, we use integers to encode timestamps and assume a total order. In general timestamps can be more complex, so that systems do not need to be globally synchronized. Associated to each timestamp is a predicate *occupyXY space* that represents an occupied box on a plane with an underlying Cartesian coordinate system. It can be seen, that over time, the first platform first moves down and than to the right. The second invariant encapsulates the downward movement of the second platform.

5.6 Window Cleaning Example: Verification Conditions

Different solutions exists to generate verification conditions from invariants with different optimization possibilities in terms of checking speed. Here, we list two:

1. A transformation takes a component invariant and unfolds the *occupyXY space* predicates as shown in Figure 7. Here, the predicate is broken down into subpredicates which are connected by a conjunction. Different possibilities for unfolding exist. The subpredicates represent the occupation of a single point on the coordinate system. If a single point is occupied, the box shall be regarded as occupied. When reducing the box to subpredicates, different occupied boxes with different coordinates become comparable and overlapping is decidable.

   After the unfolding, we only need to verify the following formula:

   $$
   \begin{aligned}
   &\forall\ t \in \{1, ..12\}\ . \\
   &\quad \exists\ occupyXY(x_1, y_1)\ ...\ occupyXY(x_2, y_2)\ . \\
   &\qquad p(I_{c_1})(t) \wedge \bar{p}(I_{c_2})(t)
   \end{aligned}
   $$

   where p is the predicate unfolding transformation and $\bar{p}$ is another unfolding transformation for indicating that an area must not be occupied. This formula can be easily split into verification conditions that can be processed by a SAT solver.

2. A second way of comparing different invariants is to take the invariants of two platforms at a time and generate for each shared timestamp a geometric verification condition characterizing the overlapping of the two boxes with edge coordinates (x_{1_c}, y_{1_c}), (x_{2_c}, y_{1_c}), (x_{1_c}, y_{2_c}), (x_{2_c}, y_{2_c}) for the a component c, e.g.,

   $$
   \begin{aligned}
   \neg\exists x, y . & x_{1_{c_1}} \leq x \leq x_{2_{c_1}} \wedge \\
   & x_{1_{c_2}} \leq x \leq x_{2_{c_2}} \wedge \\
   & y_{1_{c_2}} \leq y \leq y_{2_{c_2}} \wedge \\
   & y_{1_{c_2}} \leq y \leq y_{2_{c_2}}
   \end{aligned}
   $$

   These verification conditions can be solved by SMT solvers.

Common to both approaches is the fact that we need to compare components pairwise for each each set of shared timestamps. Additional optimization possibilities comprise the elimination of possibilities for components that do not interact with each other.

$$
\begin{aligned}
&occupyXYspace(x_1, y_1, x_2, y_2) = \\
&\quad occupyXY(x_1, y_1) \wedge occupyXY(x_1 + 0.1, y_1) \wedge ... \wedge occupyXY(x_2, y_1) \wedge \\
&\quad occupyXY(x_1, y_1 + 0.1) \wedge occupyXY(x_1 + 0.1, y_1 + 0.1) \wedge ... \wedge occupyXY(x_2, y_1 + 0.1) \wedge \\
&\qquad ... \\
&\quad occupyXY(x_1, y_2 - 0.1) \wedge occupyXY(x_1 + 0.1, y_2 - 0.1) \wedge ... \wedge occupyXY(x_2, y_2 - 0.1) \wedge \\
&\quad occupyXY(x_1, y_2) \wedge occupyXY(x_1 + 0.1, y_2) \wedge ... \wedge occupyXY(x_2, y_2)
\end{aligned}
$$

Figure 7: Predicate unfolding (conjunction of points)

5.7 Invariants and Verification Conditions in the Communicating Cars Example

Invariants in the communicating cars example are constructed in a similar fashion as in the window cleaning example. One invariant per car is constructed that indicates possible positions of a car at a distinct time. Note, that we can use disjunctions to encode non-deterministic behavior. Verification conditions are constructed with respect to the desired property. If we want to investigate communication possibilities between two cars, we use a transformation that returns true if a car is for a distinct timestamp in the range of the other car. Computing approximations of the distance can be done within standard SMT solvers or using specialized algorithms.

6 Conclusion and Future Work

We have motivated a framework for specifying and checking behavioral aspects in time, communication and space of widely distributed systems. Adapting our methodology to different application areas and investigating larger case studies remains subject to future work. Eventually we plan to model and check large systems comprising thousands of distributed components using highly parallel hardware. Model storage and checking of properties can eventually be offered as cloud services.

References

[1] M. Apel. A 3D geological information system framework. Geophysical Research Abstracts, vol. 7, European Geosciences Union, 2005.

[2] A. Basu, M. Bozga, J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. Fourth IEEE International Conference on Software Engineering and Formal Methods, IEEE 2006.

[3] B. Bennett, A. G. Cohn, F. Wolter, M. Zakharyaschev. Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. Applied Intelligence, Volume 17, Issue 3, Kluwer Academic Publishers, November 2002.

[4] S. Bensalem, A. Griesmayer, A. Legay, T. H. Nguyen, J. Sifakis, R. Yan. D-finder 2: towards efficient correctness of incremental design. In NASA Formal Methods, LNCS, Springer, 2011.

[5] 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.

[6] J. O. Blech, Y. Falcone, H. Rueß, B. Schätz. Behavioral Specification based Runtime Monitors for OSGi Services. Leveraging Applicationsof Formal Methods, Verification and Validation (ISoLA), vol. 7609 of LNCS, Springer, 2012.

[7] J. O. Blech and M. Périn. Generating Invariant-based Certificates for Embedded Systems. ACM Transactions on Embedded Computing Systems (TECS), 2012.

[8] L. Caires and L. Cardelli.A Spatial Logic for Concurrency (Part I). Information and Computation, Vol 186/2 November 2003.

[9] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). Theoretical Computer Science, 322(3) pp. 517-565, September 2004.

[10] J. Canny. The complexity of robot motion planning. MIT Press, Cambridge, 1988.

[11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement. Computer Aided Verification, vol. 1855 of LNCS, Springer, 2000.

[12] E. Clarke, O. Grumberg, D. A. Peled. Model Checking. MIT Press, 1999.

[13] B. Dutertre, L. De Moura. The yices smt solver. Tool paper at http://yices.csl.sri.com/tool-paper.pdf, 2006.

[14] S. Gottschalk , M. C. Lin , D. Manocha , S. Gottschalk , M. C. Lint , D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. Proc. ACM SIGGRAPH, 171180, 1996.

[15] D. Hirschkoff, É. Lozes, D. Sangiorgi. Minimality Results for the Spatial Logics. Foundations of Software Technology and Theoretical Computer Science, vol 2914 of LNCS, Springer, 2003.

[16] P. Jimnez, F. Thomas, C. Torras. Collision Detection Algorithms for Motion Planning. Lectures Notes in Control and Information Sciences 229, Springer, 1998.

[17] S. Kambhampati and L.S. Davis. Multiresolution path planning for mobile robots. Volume 2 , Issue: 3, Journal of Robotics and Automation, IEEE 1986.

[18] J-C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, 1991.

[19] M. C. Lin, D. Manocha. Fast interference detection between geometric models. The Visual Computer, Volume 11, Issue 10, pp 542-561, Springer, 1995.

[20] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. Formal Methods in System Design, Volume 6 Issue 1, Kluwer Academic Publishers, 1995.

[21] 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.

[22] L. De Moura, N. Bjørner. Z3: An efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems (pp. 337-340). Springer, 2008.

[23] T. Nipkow, L. C. Paulson, M. Wenzel. A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS, Springer, 2002.

[24] A. Platzer, J-D. Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). International Joint Conference on Automated Reasoning, volume 5195 of LNCS, pages 171-178. Springer, 2008.

[25] R. H. Reussner, I. H. Poernomo, H. W. Schmidt. Reasoning about Software Architectures with Contractually Specified Components. Component-Based Software Quality, vol. 2693 of LNCS, Springer, 2003.

[26] G. Smith and J. Friedman. A Technology Whose Time Has Come. Earth Observation Magazine, November 2004.

[27] A. Troynikov and H. Schmidt. Designing robust fault tolerant systems with shape. Improving Systems and Software Engineering Conference incorperating SEPG^{SM} Asia-Pacific Conference 2012, Melbourne, August 2012.

[28] C. Weaver, D. Peuquet, A. M. MacEachren. STNexus: An Integrated Database and Visualization Environment for Space-Time Information Exploitation. `http://www.geovista.psu.edu/publications/2005/Weaver_ARDA_05.pdf`, 2005.

[29] S. Dal Zilio, D. Lugiez, C. Meyssonnier. A logic you can count on. Symposium on Principles of programming languages, ACM, 2004.