Towards a Model-Based Verification Methodology for Complex Swarm Systems

(Invited Paper)

Jonas Gomes Filho[1] Nils Przigoda[1,2] Robert Wille[2,3] Rolf Drechsler[1,2]

[1]Group for Computer Architecture, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria
Email: {gomes.filho,przigoda,drechsle}@informatik.uni-bremen.de robert.wille@jku.at

***Abstract*—The recent advances with respect to the costs, size, and power consumption of electronic components paved the way for *System of Systems* (SoS), *Cyber-Physical Systems* (CPS), or the *Internet of Things* (IoT). As a next stage, these developments currently motivate the consideration of *Complex Swarm Systems* (CSS), i. e., continuously running systems that will dynamically change after deployment and are connected by heterogeneous components which can join and leave the system at any time. Due to this dynamic nature and the constant reconfigurations, it is not possible to completely verify those systems with conventional verification methods anymore. Therefore, we propose a new methodology which follows a different scheme: Instead of trying to verify all possible behavior of a CSS (which, due to the vast number of possible instantiations or connections of the heterogeneous components, becomes an impracticable task anyway), we aim for verifying that, at least, no scenario which violates certain (safety-critical) forbidden actions is possible. To this end, solutions for model-based verification are employed. By means of a case study, the feasibility and promises of the proposed methodology are illustrated.**

I. Introduction

The constant improvements in the design of circuits and systems led to more complex but also more efficient electronic systems (with respect to costs, size, and power). This eventually established the development of *System of Systems* (SoS), *Embedded Systems* (ES), and their recent extensions to *Cyber-Physical Systems* (CPS) [1], but also paved the way for the *Internet of Things* (IoT) [2] or *Swarm* systems [3].

In the latter case, a system is not constituted by single, application-specific components assembled in a clearly defined space anymore. Instead, various heterogeneous components which are globally connected in a large area may be utilized to eventually realize different applications. Moreover, these components may not explicitly be bounded to the respective systems, but may frequently enter or leave them – resulting in a highly dynamic system with frequent reconfigurations. The *Terraswarm* project [4], [5] provides several examples for such systems. In the following, we denote such systems *Complex Swarm Systems* (CSS).

However, the emerge of CSS also triggers new design challenges. This particularly holds for safety-critical systems where verification is essential in order to guarantee that a system works as expected or, at least, that the system does not cause fatal actions. For conventional embedded and cyber-physical systems, impressive improvements have been made with respect to verification in the past years [1]. However, all of them usually require a distinct formal model of the system or component to be verified. Due to the dynamic nature of CSS, such a distinct model is often not available anymore. Alternatively only checking the correctness of the single components and the single applications of a CSS is usually not sufficient to imply the correctness of the CSS and, e. g., its future reconfigurations. Hence, alternative verification methodologies have to be explored.

In this work, we discuss the resulting challenges of these systems and envision a verification methodology which proposes that every subset of components of a CSS must have a set of inherent and safety-critical *forbidden actions*. These actions must always be considered when developing new systems and applications. We also propose a new methodology for checking these forbidden actions using a model-based approach. While a complete verification of the CSS may not be possible, proving that there is no scenario where a forbidden action is executed is possible and feasible. To the best of our knowledge, this is the first proposal of a verification methodology for CSS thus far.

In the remainder of this work, these issues are illustrated and discussed by means of a running example which has also been applied to conduct first feasibility studies of the envisioned methodology. To this end, we employed verification approaches which are already established for single components as well as recently proposed solutions for systems provided in more abstract modeling languages such as UML/OCL. Because of that, the next two sections first provide a brief review of model-based verification and, afterwards, introduce the considered scenario as well as the corresponding model which is used as a running example. After that, the envisioned verification methodology is introduced in Section IV. Section V illustrates the application of the methodology to the running example. Finally, the paper is concluded in Section VI with an outlook on future works.

978-1-5090- 2541-1/16/$31.00

II. Model-based Verification

The *Unified Modeling Language* (UML) [6] together with the *Object Constraint Language* (OCL) [7] allows for modeling complex systems at an abstract level without the need to provide detailed implementations. While the general structure and behavior of the system are expressed graphically in terms of UML (class) diagrams, textual OCL constraints are used in order to add further restrictions that cannot be expressed by the UML model notation itself. The OCL is a declarative language that mainly consists of logic, arithmetic, navigation, and collection expressions. A comprehensive overview of all OCL expressions, its keywords, and the precise semantic definitions are given in [7].

UML/OCL *class diagrams* represent blueprints for a possible system. *Classes* and *associations* provide the main constructs in a class diagram. Classes describe what information can be handled within the modeled system and how the information is structured. *Attributes* define the single data elements of classes. Furthermore, a class can contain operations as well as OCL constraints for describing its behavior by means of pre- and postconditions. Finally, invariants are OCL constraints that restrict the set of valid system states by enforcing specific system properties. Overall, a model specified in terms of a class diagram can be seen as a template for creating a concrete *system state* complying to the specification.

The system states themselves are then seen as instantiations of a class diagram which, in turn, are represented by *object diagrams*. Each element in an object diagram has a corresponding counterpart in the class diagram. In other words: an object is an instantiation of a specific class holding values for each class attribute (at a particular point in time). A link connecting objects is an instantiation of an association. A system state can generally comprise any number of objects and links. A model is *consistent*, if there exists a non-empty system state which satisfies all defined OCL invariants, i. e., the system properties defined by the invariants, are not violated.

Even if UML/OCL models usually offer no precise implementation details of a complex system, this high-level of abstraction might already result in an over-constrained model such that no valid system state can be derived (inconsistent models) or in which some operations could never be executed due to too restrictive pre- and postconditions. But even if this is not the case, the specification may still allow for reaching "bad states" such as deadlocks or other unwanted behavior.

In order to detect those problems in this early stage of the design, several approaches for the validation and verification of models have been proposed. Here, the objective is, e.g., to check whether the model is consistent or not. Consistency is a typical verification task that is formally defined, e. g., in [8], [9], [10]. In order to verify the model, different approaches have been proposed, e. g., based on solvers for constraint satisfaction problems [8], [10], solvers for Boolean satisfiability [11], [12], or approaches based on relation logic [13]. Those approaches do not rely on explicitly enumerating all possible system states. Instead, they utilize a symbolic representation of the given UML/OCL model which allows to consider all possible system states. This is particularly interesting for modeling CSS, since it allows to consider all possible (re)configurations the system may assume.

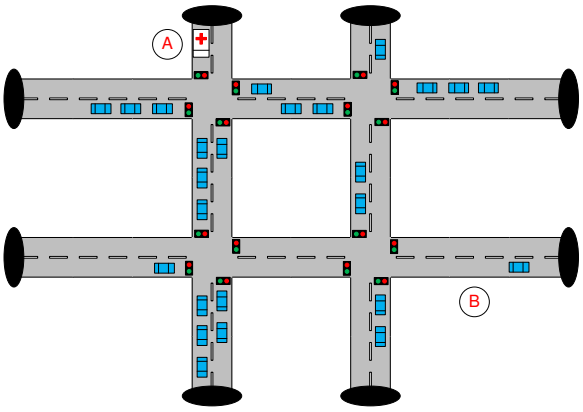


Fig. 1. CSS environment example

In this work, we rely on the solution presented in [11]. The authors propose to translate the verification task into an instance of a *Satisfiability Modulo Theories* (SMT) problem [14], [15]. Then, the problem instance can be solved using so-called SMT solvers such as proposed in [16]. These solvers allow for an efficient traversal of large search spaces and, hence, are suitable to determine whether the model description is consistent.

III. Motivational Example

A. Overview

In this work, the envisioned methodology is motivated and illustrated by means of a CSS to be designed realizing a traffic control system. This scenario is especially suited for illustrating the CSS nature because it is very dynamic, i. e., new components can join and leave the system at any time. Note, however, that the case study is not restricted to a traffic control system, but can be applied as a general methodology for CSS verification.

The recent development of technologies like *Vehicle to Vehicle communication* (V2V) as well as *Vehicle to Device communication* (V2D) – well established by the IEEE 802.11p – have raised new opportunities for the development of CSS applications [17]. Taking those technologies as a foundation, we propose a CSS composed of three types of vehicles: normal, autonomous, and emergency. They can interact with each other as well as with *Traffic Lights* (TL) and people, which are recognized by their devices using the V2D communication.

A possible scenario for the traffic control system involving the described components, is illustrated in Fig. 1. Here, four intersections and twelve streets are considered. The CSS itself is composed of components which may (dynamically) be placed throughout this environment. This, e. g., includes sensors (V2V and V2D) as well as actuators (controlling, e. g., the traffic lights and, by this, the traffic). The black ellipses at

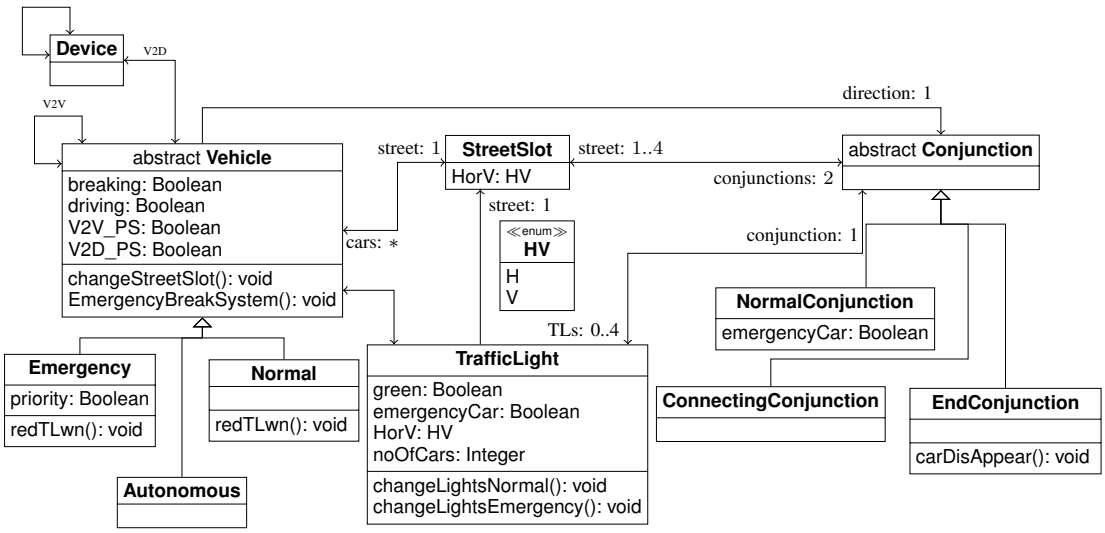


Fig. 2. Considered model

the end of each street indicate the limits of the system, i. e., vehicles can join or leave the system when crossing those points.

The resulting system can be seen as a CSS since it is composed of various single components (sensors, vehicles) which may dynamically join and leave the system, i. e., connect and/or disconnect. Moreover, on top of that system, different applications (composed of different sets of components) can be realized. In the remainder of this work, the following applications are considered:

1) A smart *TL Control Application* that operates the single lights according to the number of cars on their corresponding street, the number of wait cycles of the perpendicular TLs in the local intersection, or other metrics.
2) An *Emergency TL Control Application* which is activated when an emergency vehicle (in service) like an ambulance enters the system. As an example, consider the ambulance vehicle shown in Fig. 1 and leaving the point A for the point B. In this case, the application is supposed to detect the presence of the ambulance and, then, force the actuators to control all traffic lights on the emergency vehicle's way to green. Accordingly, the perpendicular TLs of all corresponding intersections are supposed to turn to red.
3) A *Vehicle Safety Application* which will activate an emergency break system every time a vehicle is in a eminent collision with another vehicle or a person (in order to prevent a crash or a passing over). If a vehicle tries to cross a red light, the Vehicle Safety Application will also activate the emergency breaks of all autonomous vehicles as well as set the TL to a red signal in order to warn drivers of all other vehicles.

Considering these applications, the CSS serves as a platform providing the respectively needed components. The applications in turn use a subset of the available components of the system in order to derive the required information and to trigger the respective actuators. As a traffic control system as considered here obviously is a safety-critical system, verifying its correctness is, of course, crucial. To this end, the flexibility of the system itself (components may join and leave) as well as its applications (components are used for different purposes) have to be considered. While, at the first glance, this seems to be easy to comprehend for a simple scenario as discussed here, cases causing serious problems can easily be overseen.

As a consequence, a verification methodology is required which utilizes as good as possible the existing state-of-the-art verification solutions but, additionally, considers the characteristics of the underlying CSS, i. e., its dynamic nature with numerous heterogeneous as well as joining and leaving components which may reconfigure themselves over time. In this work, we are proposing such a methodology. Applying the proposed verification scheme that will be presented in Section V eventually shows that even a presumably simple CSS as discussed here may inherit a fatal error with serious consequences.

B. Considered Model

In order to represent the interaction between the components, a structure as described in Fig. 1 is considered. For being more objective and make explanations easier, just one of the four intersections of the figure will be considered. Nevertheless, the issues discussed in this work can be easily expanded for the complete scenario and also for any combination and instantiation of the objects.

As a formal basis for these tasks, we use a description of the considered system provided in UML/OCL as shown in Fig. 2. In the remainder of the paper, this model will be used in order to illustrate the proposed solution.

The model has classes for vehicles, conjunctions, traffic lights, streets, and persons (represented by connected devices). The class *Vehicle* has an attribute for detecting when it is breaking and the attribute `driving` indicates if the car is

moving or not. V2V and V2D proximity sensors (cf. the Boolean attributes `V2V_PS` and `V2D_PS`) are set to `true` if an imminent collision to another vehicle or a person, respectively, is detected. There are three types of vehicles inherited from the abstract class *Vehicle*. The *Emergency* vehicle has an additional Boolean attribute `priority` which indicates if it is in service or not. *Normal*, i.e., non-autonomous, vehicles have an operation `redTLwn` which warns the driver if the TL is red.

Furthermore, each vehicle – independently from its precise class type – is located on one street slot and, thus, the abstract class *Vehicle* has an one-way relation to the class *StreetSlot*. Hence, streets do not provide any information; only the sensors in the vehicles. In the model, a street slot is either vertical or horizontal – defined by an corresponding enum data type. The vehicles on the streets have a direction which is one of the two related conjunctions – ensured by an omitted invariant.

Additionally, every street slot starts and ends at an abstract *Conjunction*, which can be a *NormalConjunction* for three or four streets, an *EndConjunction* for representing the end of the system (e. g., represented by the black ellipses in Fig. 1, or a *ConnectingConjunction* for connecting two slots of streets. The different relations for different directions have to be restricted until each street has exactly two ends. For modeling this, the class *StreetSlot* as well as the different conjunction classes are enriched with invariants. For example, the two defined ends of a *ConnectingConjunction* have to be different.

Furthermore, at an *EndConjunction* vehicles can appear as well as disappear – conducted by the operation `carDisAppear`. Corresponding pre- and postconditions for the operation ensure that vehicles can only disappear when they are driving towards the *EndConjunction* and are on the only connected street of the *EndConjunction*. Vice versa, for every *new* vehicle appearing on the correspondent street, its direction is set opposite to the direction of the *EndConjunction*.

The operation `changeStreetSlot` models the movement of a vehicle by changing the street slot in which the vehicle is located. If two vehicles are in the same slot, and the variable `driving` is set to true,[1] they will be dangerously close and the `emergencyBreakSystem` will be activated.

All traffic lights of a normal intersection can only change their lights either (1) in the standard mode (if no emergency vehicle driving towards this intersection is detected) or (2) in the emergency vehicle mode (if on any of the connected streets an emergency vehicle has been detected drives towards the intersection and is in service). In the latter case, the traffic light will give a signal to the intersection to ensure that all other traffic lights will switch to red. If an autonomous vehicle tries to cross a red light the `emergencyBreakSystem` will be activated. If non-autonomous vehicles intend to cross a red light, the operation `redTLwn` will be activated.

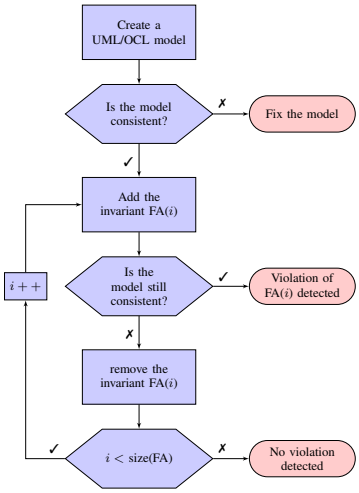


Fig. 3. Proposed methodology

IV. Proposed Methodology

Because of the reasons discussed in Section III-A, verifying a CSS is not a trivial task. In fact, checking the absence of errors of the individual components and applications is not sufficient to imply the correctness of the CSS itself. The dynamic nature of the CSS and the constant reconfigurations do not allow a completely precise model of a CSS. Therefore, it is necessary to have a new flexible and alternative verification scheme that supports the CSS.

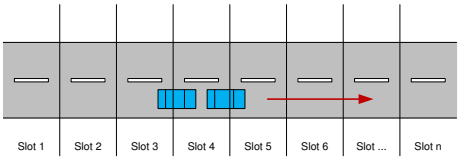
Initiatives like the Terraswarm project propose that the objects of the swarm can be released for the developers so that they are free to create applications. In a similar way, this is already done in platforms for smartphones [4], [5]. However, since those applications do usually control cyber-physical systems which can potentially deal with safety-critical systems, we propose that each subset of components of a CSS must additionally be equipped with a set of inherent forbidden actions. These forbidden actions are not allowed to occur – no matter what application or configuration the system will later assume. Hence, before releasing the components, developers have to guarantee the integrity of their CSS with respect to these forbidden actions.

For the CSS considered in this work, the following set of forbidden actions is considered:

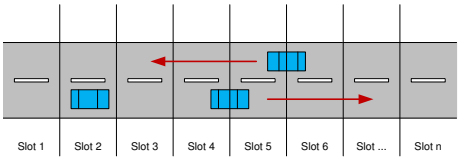
1) A vehicle invading another vehicle's safety space while it is moving.
2) A vehicle invading a person's safety space while it is moving.
3) A vertical and a horizontal TL in the same intersection being green at the same time.

The first action will prevent a direct crash involving vehicles, the second one will prevent to run over persons, and the third one prevents two or more perpendicular TLs to be green at the same time – all situations are obviously to be avoided.

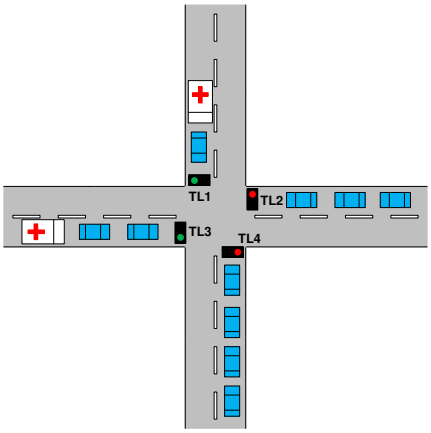
[1] Vehicles are allowed to be close if they are not in motion.



(a) Expected scenario for forbidden action 1



(b) Unexpected scenario for forbidden action 1



(c) Violation of the forbidden action 3

Fig. 4. Violation of forbidden actions

The set of forbidden actions will be then translated into OCL invariants, which will be individually checked. Fig. 3 shows the general flow of the methodology for verifying the model. FA is the set of forbidden actions with $i = 1, 2, ..., I$ being the index of this set.

First, the model must be checked for consistency as previously described in Section II. Once the model is proven to be consistent, a new class *ForbiddenAction* is added to the model. It has no relations or operations, just the currently considered forbidden action ($FA(i)$) translated into an OCL invariant. With the new invariant, the model is checked again. If it is now proven to be consistent, a violation of the FA is detected (since an instantiation of the CSS is possible where a forbidden action occurs). In this case, the model cannot be released but has to be corrected and checked for consistency again. If instead the model including $FA(i)$ is no longer consistent, it has been proven that this specific forbidden action can never occur in the CSS. This process is repeated for each FA. If the model is not consistent for each one of them, it is proven that no scenario, configuration, etc. will violate the forbidden actions.

V. Application and Results

The flow presented in Fig. 3 has been implemented and was applied in order to verify the considered model. In order to conduct the actual checks, the methodology reviewed in Section II has been utilized. More specifically, we have used the approach originally presented in [11] for the consistency checks.

In addition to that, we needed to define an upper bound for the maximal number of components (i. e., object instantiations) to be considered (required to eventually solve a finite problem). This is not a restriction to the proposed methodology as, eventually, each CSS will always be composed of a finite number of components. In our evaluations, we set this parameter to 6 objects per class, i. e., overall, the CSS can be instantiated with at most $9 \cdot 6 = 54$ components. Finally, the lower bound on the number of components is set to 1 – prohibiting the instantiation of an empty system state which always would satisfy all constraints.

Applying the methodology and this setting to the considered example and the three forbidden actions led to the following results:

The first consistency check (without considering any forbidden action) was completed successfully. This means that the model in general is free of contradictions and we can continue with checking the forbidden actions. Checking the forbidden actions, however, unveiled serious flaws. In fact, the CSS as modeled in Fig. 2 indeed allows for configurations and situations in which a forbidden action is violated. More precisely:

- The first flaw was caused by the modeling process of the movement of the car. The designer expected that the violation would occur as described in Fig. 4(a), but he has not considered that two cars are allowed to be in the same street slot as long as they are driving towards opposite directions,[2] as illustrated in Fig. 4(b). For fixing the problem, new invariants must be added for describing this situation and then checking flow must be restarted.
- The second violation was found for the third forbidden action, horizontal and vertical traffic lights are green at the same time. This forbidden action can occur, when there are two emergency cars, both in service, driving towards the same conjunction and one is located in a vertical and the other one in a horizontal street. Such a situation is sketched in Fig. 4(c). This could also happen in a scenario with three or more emergency cars, leading to a probable crash.

Both cases are not obvious and, hence, could have easily been overseen by the designers. Using the proposed methodology, it is possible to guarantee that situations or configurations in which forbidden actions occur are never possible. After

[2]In this work, we are not considering overtaking.

fixing the two mentioned flaws, the verification flow was restarted and, eventually, no further violations could have been detected. By this, the absence of forbidden actions for a CSS can be guaranteed – surely an important quality criteria for a system which dynamically may change over time.

Concerning the run time performance, the methodology is capable of proving the occurrence (or absence) of forbidden actions in acceptable time. However, in order to get a more detailed view, we further evaluated the scalability by increasing the number of components to be considered. Fig. 5 shows the resulting verification time with respect to the number of instances per class. Since there are nine (non abstract) classes of objects, the graph shows results for up to 117 objects (i. e., components). These numbers have been obtained from an Intel i5 with 2.6 GHz cores with 16 GB of RAM.

As expected, the verification time increases exponentially. This is in line with the performance of verification approaches for conventional circuits and system [1]. Hence, how to improve the verification methodology will also be an issue in the domain of CSS.

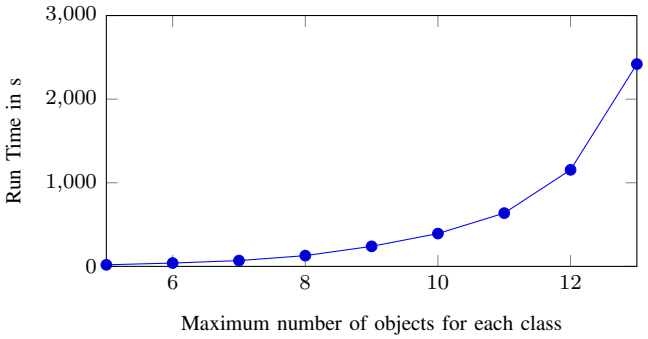


Fig. 5. Verification time for $\langle n \rangle$ rounds of sorting in an array

VI. Conclusions

The next generation of electronic systems will be subject to frequent dynamic changes after deployment and will be connected by heterogeneous components which can join and leave the system at any time. In this work, we denoted those systems Complex Swarm Systems and discussed the challenges which will emerge in their design and, in particular, in their verification. In order to address these challenges, we envisioned a verification methodology which is capable of checking the considered systems despite their dynamic nature. To this end, we do not aim for a complete verification anymore, but, instead, guarantee that at least no forbidden actions can occur. Solutions for model-based verification have been applied for this purpose. By means of a running example, the underlying problem and the feasibility of the proposed methdology has been illustrated and demonstrated, respectively. By this, we are laying down the foundation for the verification of CSS. Future work will focus on improving the proposed methodology with respect to efficiency and scalability.

VII. Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project SPECifIC under grant no. 01IW13001, the German Research Foundation (DFG) within the Reinhart Koselleck project under grant no. DR 287/23-1, the Graduate School SyDe funded by the German Excellence Initiative within the University of Bremen's institutional strategy, Siemens AG, and the Brazilian program Science Without Borders, through the National Council for Scientific and Technological Development (CNPq).

References

[1] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2011.

[2] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[3] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber, "A vision of swarmlets," *Internet Computing, IEEE*, vol. 19, no. 2, pp. 20–28, 2015.

[4] E. A. Lee, J. D. Kubiatowicz, J. M. Rabaey, A. L. Sangiovanni-Vincentelli, S. A. Seshia, J. Wawrzynek, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, R. Jafari, D. Jones, V. Kumar, R. Murray, G. Pappas, A. Rowe, C. M. Sechen, T. S. Rosing, B. Taskar, and D. Wessel, "The TerraSwarm Research Center (TSRC) (A White Paper)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-207, Nov 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-207.html

[5] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. S. Rosing, J. Wawrzynek, D. Wessel, J. M. Rabaey, K. Pister, A. L. Sangiovanni-Vincentelli, S. A. Seshia *et al.*, "The swarm at the edge of the cloud." *IEEE Design & Test*, vol. 31, no. 3, pp. 8–20, 2014.

[6] J. Rumbaugh, I. Jacobson, and G. Booch, Eds., *The Unified Modeling Language reference manual*. Essex, UK: Addison-Wesley Longman Ltd., 1999.

[7] OMG – Object Management Group, "Object Constraint Language," 2014, version 2.4, February 2014. [Online]. Available: http://www.omg.org/spec/OCL/2.4

[8] J. Cabot, R. Clarisó, and D. Riera, "Verification of UML/OCL Class Diagrams using Constraint Programming," in *ICST Workshops*, 2008, pp. 73–80.

[9] M. Gogolla, M. Kuhlmann, and L. Hamann, "Consistency, Independence and Consequences in UML and OCL Models," in *Tests and Proofs*, 2009, pp. 90–104.

[10] J. Cabot, R. Clarisó, and D. Riera, "On the verification of UML/OCL class diagrams using constraint programming," *Journal of Systems and Software*, vol. 93, pp. 1–23, 2014.

[11] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, "Verifying UML/OCL models using Boolean satisfiability," in *Design, Automation and Test in Europe*. IEEE, 2010, pp. 1341–1344.

[12] M. Soeken, R. Wille, and R. Drechsler, "Encoding OCL data types for SAT-based verification of UML/OCL models," in *Tests and Proofs*, 2011, pp. 152–170.

[13] M. Kuhlmann and M. Gogolla, "From UML and OCL to Relational Logic and Back," in *Int'l Conf. on Model Driven Engineering Languages and Systems*, 2012, pp. 415–431.

[14] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," 2016. [Online]. Available: http://www.SMT-LIB.org

[15] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories." *Handbook of satisfiability*, vol. 185, pp. 825–885, 2009.

[16] L. M. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for Construction and Analysis of Systems*, 2008, pp. 337–340.

[17] A. Khelil and D. Soldani, "On the suitability of device-to-device communications for road traffic safety," in *IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 224–229.