Law-based Verification for Complex Swarm Systems

(Extended Abstract)

Rolf Drechsler*† Hoang M. Le* Mathias Soeken*† Robert Wille*†

*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{drechsle,hle,msoeken,rwille}@informatik.uni-bremen.de

I. Introduction

Motivated by the recent advances with respect to the costs, size, and power consumption of electrical and sensor systems, the design of swarm systems received significant attention in the past decade. While several manifestations are investigated (including the "classical" *System of Systems* (SoS), their recent extensions to *Cyber-Physical Systems* (CPS), as well as networks such as proposed in TerraSwarm [1] and the *Internet of Things*), swarm system are commonly characterized by a large number of heterogeneous components which dynamically change their structure [1] — in the following such a system is referred to as *Complex Swarm System* (CSS). These properties pose a daunting challenge to the development of methods and solutions ensuring the correctness of these systems, i.e. to methods that check whether a given CSS indeed behaves as intended [1].

In this extended abstract, we discuss the corresponding verification challenges of these systems. In fact, checking the correctness of the components and applications of a CSS is usually not sufficient to imply the correctness of the CSS itself. Furthermore, since the functionality of a CSS is not specifically predetermined, a precise model is usually not available for verification. In order to address these issues, we propose a verification scheme that does not rely on a complete formal representation of the structure and the functionality of a CSS, but rather focuses on the verification of a set of *laws* prohibiting the components in the CSS from performing illegal actions. Based on these laws, a verification methodology is envisioned that does not require a precisely specified model anymore. The key issues of this methodology as well as resulting open research questions are discussed in this paper.

II. Verification of CSS

In the remainder of the paper we refer to a *Complex Swarm System* (CSS) as a continuously running system that is dynamically changing and operates due to individual *components* which can join and leave the CSS at any time. The CSS itself does not have a specifically predetermined functionality except from serving as a platform for *applications*. The applications in turn do serve a purpose for which they are using a subset of the components in the CSS.

This implicit hierarchy leads to three levels of verification:

1) *Each component* has to be verified on its own which ensures that a component functions according to its specification. This can be performed by established methods for (formal) verification such as assume-guarantee reasoning [2] or property checking [3].
2) *A CSS application* has to be verified in order to ensure that it follows a well-defined specific functionality. Since an application can just be considered as a conventional system of systems with respective subsystems and subcomponents, established compositional verification techniques can be applied for this purpose.
3) Finally, *the CSS* itself needs to be verified as the correctness of its components and its applications does not necessarily imply the correctness of the CSS. To the best of our knowledge, no verification technique exists for this purpose thus far.

Obviously, the most challenging question is how to conduct verification of the CSS itself. Since the CSS itself does not serve a predetermined specific functionality, the verification task considers global properties of the CSS such as integrity which should not be harmed by any unforeseen interplay of its components. We refer to these global properties as *laws* which need to be obeyed by each component and application in order to ensure the well-behavior of the CSS. Furthermore, ensuring a fully verified CSS is likely to be impracticable due to its immense complexity. As a consequence, we suggest an approach in which the correctness of a CSS must be guaranteed only with respect to its safety-critical applications. Possible directions and open questions on how to realize such a verification scheme for a CSS are discussed next.

III. Law-based Verification for CSS

As mentioned above, one of the main challenges for the verification of a CSS is the absence of a precise model of the system. A similar and related concept has recently been investigated in the context of hardware and software verification. Here, verification techniques have been lifted into higher levels of abstraction considering verification tasks on formal models of a specification (see e.g. [4]). For this purpose, modeling languages such as UML and SysML have been utilized. This enabled to formally prove general and conceptual questions about the specification in the absence of an implementation.

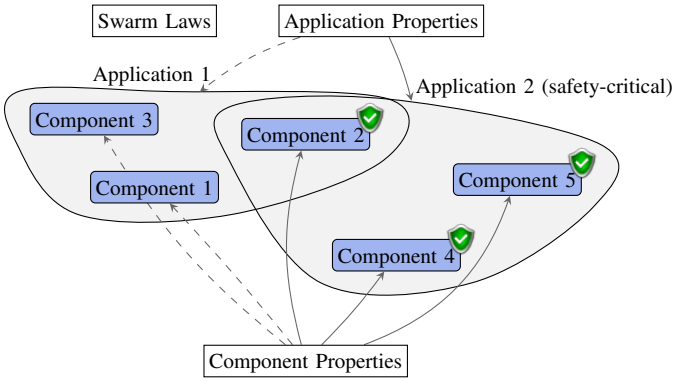

Fig. 1. Law-based Verification of Swarm Systems

Taking this idea, a CSS can also be verified by lifting the respective descriptions into higher, more abstract, levels. Then, methods as e.g. proposed in [5] can similarly be applied to prove conceptual properties on an abstract model of the CSS's environment. However, the application of these schemes to CSS verification is not obvious and hindered by several open research questions of which three are discussed in the following. Furthermore, initial ideas for solving the questions are given.

How to describe laws for incompletely specified environments?

How to formalize laws is not obvious since e.g. an actual configuration or a current state of the CSS, i.e. its environment, cannot be predicted. As mentioned above, abstract representations might be a solution. Instead of specifying precise functionality, laws may just enforce that a component in a system cannot make another component to behave against its intended functionality. Well-defined protocols or contracts can be used for this purpose (see e.g. [1], [6], [7]). These often already exist, since the components need a mechanism to communicate with each other, e.g. using the *hypertext transfer protocol*. However, currently used protocols are often either too specific (which does not allow to predict unforeseen behavior) or too general (which makes them hard to verify).

How to make verification feasible?

In fact, in an ever-changing system completeness can hardly be achieved and is not even of major interest. Often, CSS functionality covers tasks such as information-retrieval, simple communication, or monitoring which is not necessarily safety-critical. Following the argumentation from [1], we propose to distinguish between *trusted* and *untrusted components*, where trusted components obey a special consideration. Untrusted components are not allowed to be part of any safety-critical application and therefore do not need to be fully verified. On the contrary, trusted components need to be fully verified with respect to their functionality but also with respect to the CSS laws. Hence, correctness needs to be guaranteed throughout all levels of verification for which completeness-driven techniques can be applied (see e.g. [8]). This enables trusted components to be part of a safety-critical application.

Fig. 1 sketches an example excerpt from a CSS with two untrusted and three trusted components. A safety-critical application can be built by only making use of trusted components. The figure also emphasizes the three levels of verification. While the CSS laws are valid for the overall CSS, application properties only need to be shown for Application 2 and are optional for Application 1, indicated by the dashed line.

How to trust an adaptive system?

As the behavior of a CSS and the context-specific function of each component may change over the time, pre-fabrication verification is not sufficient for the trusted components. Hence, in order to make these adaptive CSS trustworthy, all trusted components need to be enriched with a *self-verification* ability. One possible solution to realize that is to use *monitors* which check for every action whether it violates a law or not. Such an approach is easy to implement but also slows down the overall speed of operation. Another more scalable approach performs verification directly on the component [9] after a change has been made, e.g. by running a model checker inside the CSS thereby exploiting its real-time capabilities.

Consider that Component 2 is adaptive in Fig. 1. If it changes it needs to be verified in order to keep its status as a trusted component. If it fails the verification, Application 2 is no longer a safety-critical application, however, Application 1 can still continue to operate since it does not require to be safety-critical.

IV. Conclusions

This extended abstract envisions a verification methodology for *Complex Swarm Systems* (CSS). As key issue, the proposed methodology focuses on a set of laws instead of a precise model of the functionality. Open research questions and initial ideas how to address them have been discussed.

References

[1] E. A. Lee *et al.*, "The TerraSwarm research center (TSRC) (A white paper)," University of California at Berkeley, Tech. Rep. UCB/EECS-2012-207, 2012.

[2] C. B. Jones, "Specification and design of (parallel) programs," in *IFIP Congress*, 1983, pp. 321–332.

[3] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science*, 1977, pp. 46–57.

[4] R. Drechsler, M. Soeken, and R. Wille, "Formal Specification Level: Towards verification-driven design based on natural language processing," in *Forum on Specification and Design Languages*, 2012, pp. 53–58.

[5] 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*, 2010, pp. 1341–1344.

[6] V. Cortier, "Verification of security protocols," in *Verification, Model Checking, and Abstract Interpretation*, 2009, pp. 5–13.

[7] M. Soeken, R. Wille, and R. Drechsler, "Verifying dynamic aspects of UML models," in *Design, Automation and Test in Europe*, 2011, pp. 1077–1082.

[8] R. Drechsler, M. Diepenbeck, D. Große, U. Kühne, H. M. Le, J. Seiter, M. Soeken, and R. Wille, "Completeness-driven development," in *Graph Transformations*, 2012, pp. 38–50.

[9] J. Barnat, L. Brim, M. Ceska, and T. Lamr, "CUDA accelerated LTL model checking," in *Parallel and Distributed Systems*, 2009, pp. 34–41.