Verification and Uncertainties in Self-integrating System

Lukas Esterle
Aarhus University
DIGIT
email: lukas.esterle@ece.au.dk

Barry Porter
School of Computing and Communications
Lancaster University
email: b.f.porter@lancaster.ac.uk

Jim Woodcock
University of York &
Aarhus University
email: jim.woodcock@york.ac.uk

***Abstract*—Self-integrating and self-improving system are required to verify their state in order to understand whether they have achieved their goal or need to adapt themselves to reach it. In this short position paper, we outline the main challenges specifically when verifying systems interacting with each other and operating under uncertainties. A short outline of the uncertainties is given as well as a brief roadmap to overcome the main challenges faced by autonomous interwoven systems, operating in an open world with incomplete knowledge.**

***Index Terms*—self-integrating systems, verification, self-improving, systems-of-systems, self-assembling**

I. Introduction

Computing systems, particularly Cyber-physical systems, are constantly interacting with their environment. In order to operate efficiently these systems need to integrate themselves properly and improve over time. This can be achieved by optimising their individual activities but also by combining their efforts in achieving their individual and common goals [1]. Along with their own adaptation, the environment may change and adapt accordingly, further increasing the challenges of the integration and improvement process. To tackle this problem, each system is required to be aware of its capabilities, deal with the uncertainty of deployment environments that change over time, and check their state and performance during runtime [2]. Many concepts which support these core mechanics of self-integration and self-improvement have been well explored by the community.

However, *verification* for these kinds of systems has received very little attention by comparison. Verification provides guarantees about the behaviour of a deployed system. For self-integrating systems it is a concern both for each individual system and for the collective of systems. To perform verification of self-integration systems, a range of different flavours of formal verification techniques is available, such as model checking, theorem proving, co-simulation, runtime monitoring, testing, and managing system assurance cases in combination with digital twins. Our vision for system verification is to bring a set of these techniques together into a sound toolchain to provide a novel semantic framework for unifying best practice methods, models, and formalisms [3], [4]. Each tool will use notations and verification algorithms most suited to their application, but there will be sound and practical interfaces between the tools to permit exchange and composition of results. Bounded verification will be carried out using any of the tools in the chain and these will be generalised by proving theorems for larger bounds and even unbounded systems. The added expense of theorem proving will be justified by overcoming the state-explosion problem. The toolchain will include humans, software, hardware, components, systems, and environments in the loop, as necessary. In this short position paper, we discuss verification of SISSY systems. We identify the main challenges in verifying these systems using two different examples: emergent software systems and self-assembling robots.

II. Self-Integrating Exemplars

Emergent software systems [5] are assembled from a large pool of potential building blocks. Many building blocks have a set of potential implementation variants. These could include different scheduling policies or cache eviction strategies. The functionality of each building block can be verified in isolation. Real-time learning determines in deployment which combination of building blocks best suits each environment encountered by the system. This could be the different request patterns in a data centre, for example. System design is therefore an emergent property relative to the current environment. The deployed system may undergo hot-swaps of individual logic units and entire architectural sub-areas during execution. Emergent software systems exist on compute nodes. This could be a single compute node built from individual software blocks. It could also be on multiple compute nodes with multiple systems interacting to provide a common service. The individual systems could be a load balancer, a replicated collection of web servers, a cache layer, and a database. The decision-making on how each system is composed, and how the distributed graph of interconnected systems is formed, may be centrally orchestrated or may rely on decentralized local controllers. In such systems, there is usually a set of capabilities that must always be present and a set of capabilities that can be used if available. In the data-centre example, at least one web server and one database instance would represent a minimal viable system, while the presence of a load balancer, replicated web servers, and a cache layer, can all provide added utility if available.

Self-assembling robots are autonomous devices, able to interact with the environment through sensors and actuators and contain the required control software locally. They can operate with other systems and change the environment in the absence of a central controller. Self-assembly itself is observed in nature with social insects (e.g., ants and termites) creating different types of physical structures. Here individuals combine their competencies to achieve a common, collective goal through coordination and/or self-organisation. Often, this goal is to establish a specific structure, such as a bridge, raft, or tower. Self-assembling robots create a specified structure through self-organisation [6], [7]. In contrast to our emergent software system example, each robot may be able to provide some useful function in isolation, without the presence of any other robots, but may have value-added capabilities when other robots are encountered. We can consider robots performing search and rescue operations. Individual robots can search for victims, together, multiple robots can transport the victim or create an autonomous communication network to alarm human rescue personnel [8]–[10].

Figure 1 contains a specific scenario with self-assembling robots. The four robots in green belong to the same stakeholder, with an intended structure shown at the bottom. The robots can move in two dimensions on the ground. They can rotate and reorient themselves in place. Over time, the robots will approach each other (b) and start to connect. A new robot (in red) is coming close and joining the structure at time (c), achieving the required structure.

In this example, the individual robots have to be able to localise themselves in the environment as well as relative to the other robots. They can use their sensors or mutual information exchange leading to absolute or at least relative positioning. The three robots have to be able to understand that they are connected. In the best case, they also know that they have completed the task with the external red robot and constructed the desired pattern. In the worst case, the green robots operate under the assumption that they have not completed the task as there are only three connected robots. In this case, situation (e) could occur, where the fourth robot will add itself to the existing structure. If the intended structure needs to be oriented in a certain way, the collective of robots now has to reorient itself, possibly including the external red robot.

III. Uncertainty and Verification

Based on our two examples, we provide an initial analysis of the verification space for self-integrating systems that operate under uncertainty.

Verification can assure that a deployed system satisfies its expected requirements. In practice, this can be achieved at either (or both) development time or runtime. Verification has classically been achieved at development time, using a mixture of static analysis and execution testing. Both phases are typically performed during development and before deployment. In this classical sense verification problems are usually solved by dividing the problem into what can reasonably be solved by static assertions, and what must be solved by dynamic tests. Static assertions are often underpinned by language design theorems such as static typing. They reduce how a system can go wrong during execution. These assertions tend to reduce the volume of unit tests needed to assure well-defined behaviour.

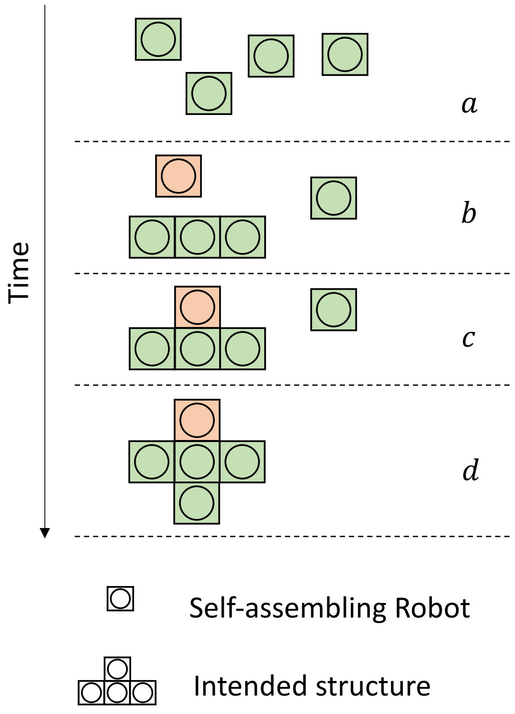

Fig. 1. Self-assembling robots at different time-steps a, b, c, d. Four green robots are intending to generate a specified structure. The red robot joins the environment (step b) and integrates with three of the green robots (step c).

Over the last decade, the concept of runtime verification [11] has gained significant interest as an additional tool in the verification kit. Runtime verification can be used when certain information about a system's deployment environment is not known in advance such that a test suite could not represent complete coverage of a system; in these cases runtime verification uses injected monitors to determine whether the actual state of a system within a finite execution period matches the expected state—and specification—of the system according to its current execution conditions.

To perform verification, either *a priori* or during runtime, *digital twins* can be utilised. A digital twin for a CPS aims to represent the physical system as closely as possible and includes a simulation model that allows prediction of how a physical system should behave [12]. Sensor and actuation signals are streamed from the CPS (the physical twin) to the multi-model co- simulation (the digital twin). The digital twin can then predict how the physical twin should behave. However, the predictions can never be 100% accurate. This is because the multi-models cannot capture the full detail of physical reality: the sensor data is noisy and sampled at a finite frequency; the numerical solution of the multi-models is an approximation; and the environment for the physical twin may be different from the one used for prototyping. This introduces sources of uncertainty. One approach to handling this is to quantify and characterise violation events for a given safety property for the physical system. The digital twin can use a runtime monitor to check whether the noise and violations observed fall within expected statistical distributions. It can then take corrective action: to tune the multi-models or to modify the physical system's behaviour.

Runtime verification can be used to measure discrepancies between a system and its digital twin. Signal Temporal Logic (STL) [13], an extension of LTL, is a popular language for specifying dense-time, real-valued signal properties with quantitative timing constraints. Efficient algorithms exist for checking STL specifications during runtime verification. For example, Maler and Nickovic provide a simple efficient algorithm for checking satisfaction for the future fragment of STL by backward interval marking [13]–[15]. Extending STL to cope with discrepancies is a significant theoretical challenge [16]. It also raises a number of practical challenges. For example, the statistical analysis required to check discrepancies can be computationally expensive, even though the simulations needed may be run in parallel. The analysis requires noise in physical devices, actuators, and sensors to be appropriately quantified. The standard deviations for noisy signal can be obtained from empirical data and requires sound physical experimentation with the real system and its components.

We conjecture that self-improving and self-integrating systems involve two specific challenge dimensions to verification. The first is that the way in which system elements may interact is difficult to predict pre-deployment. Because the set of possible interactions may be very large (or may be infinite), exhaustively testing all of them may be prohibitively expensive or impossible to test. The second is that the deployment environment itself may exhibit properties that are hard to capture and model in advance. The set of all possible deployment environment conditions are also therefore difficult to test before deployment.

This leads us to our two exemplars of self-integrating and self-improving systems in the light of classical verification and runtime verification. We can argue that emergent software is mostly cast as a good place to classical verification in how to control state machine coherence between interacting elements during design time. While self-assembly systems are also subject to verification at design time, the state-space explosion coming about the highly dynamic environment further requires verification at runtime. This allows to verify states that might not have been covered during design-time verification. We note that self-integrating systems – particularly those which are distributed – entail a wide range of other challenges, such as partial failures. In this paper we specifically focus only on those elements that relate to the actual integration of systems rather than other non-functional concerns.

A. Unplanned interaction

We define unplanned interaction broadly as the situation in which two system elements interact in a way that could not be predicted or tested in advance.

A foundation for emergent software systems is the ability to hot-swap logic while the system is running. This should be possible without any observable interruption to ongoing service and without leaving the system in an undefined state. Although each building block has a well-defined and statically typed interface, the internal state machines of different building blocks may not necessarily be in a mutually compatible state when introduced to one another at an arbitrary time during execution [17]. Consider a system formed from three building blocks a, b, c, where a drives state transitions in b and c. At time t we can hot-swap a to an alternative implementation a' but keep b and c in the system as their implementations have not changed. In this scenario, we are relying on whatever states b and c hold being acceptable to the test envelope of a'. If this is not the case, then we have a system in an untested set of collective states. This is equivalent to undefined behaviour about which we can make no assertions. Hot-swaps at time t_i and t_j may result in different sets of states. In general, this requires an infinite set of test cases, and so is untestable. Solving this problem relies on using a programming model that can guarantee mutual compatibility of state machines between different system elements that are introduced to one another at arbitrary times.

In self-assembling robots, we encounter a similar situation when behaviour or structures emerge during runtime. Illustrated in Figure 1, a red robot joins the environment and creates the desired structure in step c already. In this case collective of robots is now relying on a component that was not designed for the task at hand by the developer of the green robots. This does not necessarily mean that the structure is not already achieved and complying with all required properties. However, similar to emergent software systems, this new, previously unexpected state is untested and unverified and it is unclear if this unexpected component complies with the requirements of the emergent structure. Solving this from a structural viewpoint, the collective of robots need to be able to form an consensus or utilise an external evaluator, able to perceive the entire structure. The external evaluator might be able to verify some of the global parameters (e.g. the shape of the structure). Verifying internal (e.g. required stability of each individual robot) and intra-robot properties (e.g. required stability across connected robots) will be more challenging and relies on the information provided by the respective components.

B. Unplanned uncertainty

We define unplanned uncertainty as the potential for environments about which we have limited knowledge before deployment. Such uncertainty can be further classified as epistemic uncertainty (insufficient knowledge) and aleatoric uncertainty (environments with inherent randomness).

In emergent software systems, epistemic uncertainty arises in how machine learning attempts to classify the environment into discrete classes that are of high utility to online reinforcement learning. Here we can have uncertainty in two different ways: uncertainty over what characteristics of the environment matter to the performance of the system; and uncertainty over which compositions of a system are likely to perform best in each environment. In the former category, for a generic classifier deployed against an unknown set of building blocks in an unknown environment, generic environment information such as request names must be translated into higher-level classes such as 'cacheability of stream' that are relevant to how the overall system performs. In the latter category, unless all the environment classes are known in advance, determining which composition of build blocks works best in each environment must be performed at runtime as and when those environments occur — when the longevity and stability of each such environment may challenge effective learning.

Self-assembling robots, operating in changing environmental conditions are constantly prone to aleatoric uncertainties. Rapidly unfolding and previously unexperienced situations cannot be learned and systems can only respond to the best of their understanding of the situation. Composing homogeneous robots into a specific shape does not need to take the order and position of each robot into consideration as they are interchangeable and the solution space is mostly affected by the aleatoric uncertainties of the environment. However, with heterogeneous robots, the position of each robot becomes relevant and may affect the final structure, adding an additional dimension to the potential solution space. Furthermore, when the location of the assembled structure is variable, having sufficient information about the environment allows the robots to select the most suitable location to build the structure. Having a heterogeneous set of robots can provide an advantage when different types of robots bring about different benefits, as well as drawbacks, depending on the environment. Unknown environmental conditions will require machine learning techniques to explore and discover new knowledge and later exploit the gained knowledge. Verifying the best solution becomes hard if the situation remains rare and does not allow for learning of alternative solutions over time.

The main challenges to verification here are (i) environmental characteristics that break the assumptions against which compositions were tested; and (ii) environmental volatility under learning, which presents a risk to the length of time for which a system will remain in explorative learning more versus exploiting learned information – where such timings would have been assumed to fall within certain bounds for effective system operation.

IV. Challenges and Roadmap

Supporting verification of self-integrating and self-improving systems opens up several new challenges. In the following we give a rough outline of the identified challenges and how they relate to the verification process and the underlying goal of the verification. We scope our challenges to those that specifically relate to self-integration, rather than to tangential and more general issues such as fault tolerance.

Information exchange among individual systems about their own state, their abilities, and their observed environmental conditions needs to be facilitated. Protocols and standards will help to alleviate this problem, however, this requires all systems, even those deployed by other developers, to adhere to them. Furthermore, information exchange needs to be technically possible. That would include established communication links and common communication technologies. Systems might initially not be aware of their own abilities to communicate with others [18] and the capabilities of others in their environment. This requires them to be able to explore approaches to interact and communicate with each other [19]. Systems belonging to the same owner might not encounter the problem of knowing and communicating their abilities. Communication protocols are in most cases clear and exploration of such aspects among those systems may not be required. With a new system (such as the red one in our Fig. 1), this might be different as its abilities might be unknown. Finally, information exchange among two or more systems brings about challenges well explored from distributed systems such as broken communication links [20] or dealing with quiescence [21].

Semantic alignment of the exchanged information is required by all participants that need to be verified. Enabling systems to establish semantics through *play* and *controlled experimentation*, can alleviate the need for dedicated domain models and ontologies. Now such mutual play or controlled experimentation brings about a challenge of enabling systems to engage in this mutual play to explore their and align their semantics. Systems need to explore semantics without compromising the safety as well as security of themselves and other participating systems. All participants are required to follow specific rules which need to be defined and agreed upon *a priori* [22], [23]. As in the previous challenge, systems belonging to the same owner are most likely able to understand each other's semantics. Systems belonging to different owners however might not be able to comprehend and understand each other.

Consensus approaches may be useful as first-class entities to help reason about state machine alignment and integration between different parts of a system or across all systems. Naturally this brings about additional challenges faced in consensus finding algorithms. Achieving a consensus on the position of a state machine involves both aspects (a) to model relevant parts of a system as generic consensus problems, and (b) to develop and select suitable centralised or decentralised consensus algorithms to solve those problems at runtime. However, finding a consensus on the current position of the state machine allows them collectively drive it towards a mutually verified holistic position. With systems coming from a different owner, different consensus algorithms might be utilised. Solving the first two challenges might help to overcome this problem with systems belonging to a different owner. Another aspect is that everyone needs to trust the input to the consensus finding algorithm - this is further highlighted with an additional challenges below.

Collaborative learning will support systems to tackle epistemic and aleatoric uncertainties by combining their knowledge and covering a larger exploration space more quickly. Approaches to coordinate the learning as well as aggregating the knowledge afterwards are needed to tackle this challenge. This leads again to challenges also arising from consensus finding. One technique could be federated learning using deep neural networks [24]. However, this introduces an overhead in generating and processing neural networks giving rise to a trade-off between utilised and available resources and the benefit all systems can get from utilising the collaborative learning approach. For such autonomous systems with limited processing power, it will be important to develop simple learning techniques that are as efficient in combining learned information as federated learning mechanisms. When systems belong to different owners they might utilise different and incompatible learning approaches – for example the red and green systems in our robotics example may both be able to utilise collaborative learning, but due to their different learning approaches may not be directly compatible with each other. It is therefore important to ensure that we avoid combinations of incompatible learning outcomes.

Model calibration allows a system to adapt an established model to better fit a new context. This may aid a system in dealing with aleatoric and epistemic uncertainties by introducing and refining probabilistic parameters in a given model based on the observed environmental conditions. By adapting models on an ongoing basis, we expect systems to be able to deal with aleatoric uncertainties due to the increased amount of data utilised to build the model. This should allow them to verify their current states more accurately. To deal with epistemic uncertainties, more data from the same source will most likely not be of help. However, model calibration can also be performed by adding new information to the model originating from new sources. These new sources can be newly discovered capabilities of an individual system or emerging from an entire collective. In the best situations, all involved systems will share models and calibrate them collaboratively. Mechanisms and techniques to do such collaborative calibration and consequently utilising such collaborative models are currently missing.

Trust among systems operating in a shared environment is necessary, specifically when the systems are developed and deployed by different developers. Relying on information from malign systems can lead to devastating results in self-integration processes. Trust can be a shortcut to reduce verification costs and can be gained by verifying outcomes rather than specific data [25]. Utilising probabilistic reasoning techniques can be one way forward to tackle this problem. Additionally, there are several models of trust that allow for systems to collaborate on establishing trust among heterogeneous and diverse members [26], [27]. Utilising such trust models as a tool to improve collaboration and verification in self-integrating systems requires those models to have a minimal processing overhead in order to be useful.

Verification envelope distance functions may be needed to describe the extent to which different potential integrations have been verified to be correct under different environments. Such an approach would allow a system to combine the verification results from individuals into a common, more coherent verification status for an entire collective system. Nevertheless, a conversion mechanism from the individual states of single systems to the combined state of the collective is required as the collective state might be the result of emergence rather then the simple sum of the individual states. If this can be achieved, such verified intermediate states would be useful to allow reasoning over how to drive a system towards an available integration which has a higher verification confidence under the currently observed environment conditions. To make decisions on the actions of individual system from an emerged collective state also requires the inverse conversion mechanism potentially requiring an understanding of causal relations of interactions initially leading to the current state.

V. Summary

In this paper we have highlighted to lack of research to date in *verification* for self-integrating systems that operate under uncertainty. Considering two motivating examples, of emergent software systems and self-assembling robots, we have derived a set of verification challenges and proposed a roadmap of initial research challenges for the community to consider. We hope that these challenges will serve as a useful discussion point for the SISSY community and will motivate further investigation of these important topics in the future.

Acknowledgment

We are grateful to the Poul Due Jensen Foundation, which has supported the establishment of a new Centre for Digital Twin Technology at Aarhus University and supported parts of this research financially. This work was also partly supported by the UK Leverhulme Trust via the Self-Aware Datacentre project, grant RPG-2017-166.

References

[1] K. Bellman, S. Tomforde, and R. P. Würtz, "Interwoven systems: Self-improving systems integration," in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. USA: IEEE Computer Society, 2014, pp. 123–127.

[2] K. Bellman, J. Botev, A. Diaconescu, L. Esterle, C. Gruhl, C. Landauer, P. R. Lewis, P. R. Nelson, E. Pournaras, A. Stein *et al.*, "Self-improving system integration: Mastering continuous change," *Future Generation Computer Systems*, vol. 117, pp. 29–46, 2021.

[3] M. Gleirscher, S. Foster, and J. Woodcock, "New opportunities for integrated formal methods," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 117:1–117:36, 2020.

[4] S. Foster, J. Baxter, A. Cavalcanti, J. Woodcock, and F. Zeyda, "Unifying semantic foundations for automated verification tools in isabelle/utp," *Sci. Comput. Program.*, vol. 197, p. 102510, 2020.

[5] B. Porter, M. Grieves, R. Rodrigues Filho, and D. Leslie, "RE^X: A development platform and online learning approach for runtime emergent software systems," in *Symposium on Operating Systems Design and Implementation*. USENIX, November 2016, pp. 333–348.

[6] E. Sahin, T. H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo, "Swarm-bot: Pattern formation in a swarm of self-assembling mobile robots," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 4. IEEE, 2002, pp. 6–pp.

[7] L. Murray, J. Timmis, and A. Tyrrell, "Modular self-assembling and self-reconfiguring e-pucks," *Swarm Intelligence*, vol. 7, no. 2, pp. 83–113, 2013.

[8] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada, "Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, vol. 6. IEEE, 1999, pp. 739–743.

[9] S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, D. Laneri, L. Winkler, T. Schmickl, R. Thenius, P. Corradi, and L. Ricotti, "Symbiotic robot organisms: Replicator and symbrion projects," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, ser. PerMIS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 62–69.

[10] M. Read, C. Möslinger, T. Dipper, D. Kengyel, J. Hilder, R. Thenius, A. Tyrrell, J. Timmis, and T. Schmickl, "Profiling underwater swarm robotic shoaling performance using simulation," in *Conference Towards Autonomous Robotic Systems*. Springer, 2013, pp. 404–416.

[11] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[12] L. Esterle, C. Gomes, M. Frasheri, H. Ejersbo, S. Tomforde, and P. G. Larsen, "Digital twins for collaboration and self-integration," in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 2021.

[13] O. Maler, D. Nickovic, and A. Pnueli, "Checking temporal properties of discrete, timed and continuous behaviors," in *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, ser. Lecture Notes in Computer Science, A. Avron, N. Dershowitz, and A. Rabinovich, Eds., vol. 4800. Springer, 2008, pp. 475–505.

[14] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, Sept. 22–24, 2004, Proceedings*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Springer, 2004, pp. 152–166.

[15] D. Nickovic, "Checking timed and hybrid properties: Theory and applications," PhD thesis, Université Joseph Fourier, Grenoble, 2008.

[16] J. Woodcock, C. Gomes, H. D. Macedo, and P. G. Larsen, "Uncertainty quantification and runtime monitoring using environment-aware digital twins," in *Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends*, T. Margaria and B. Steffen, Eds. Springer International Publishing, 2021, pp. 72–87.

[17] B. Porter and R. Rodrigues Filho, "A programming language for sound self-adaptive systems," in *International Conference on Autonomic Computing and Self-Organizing Systems*. IEEE, September 2021.

[18] L. Esterle and J. N. Brown, "The competence awareness window: Knowing what i can and cannot do," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 2020, pp. 62–63.

[19] K. Bellman, "Position paper: Towards a method for characterizing and improving integration among different systems," in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 2021.

[20] S.-Y. Tu and A. H. Sayed, "Mobile adaptive networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 649–664, 2011.

[21] M. K. Aguilera, W. Chen, and S. Toueg, "Heartbeat: A timeout-free failure detector for quiescent reliable communication," in *International Workshop on Distributed Algorithms*. Springer, 1997, pp. 126–140.

[22] J. G. Carbonell and Y. Gil, "Learning by experimentation," in *Proceedings of the Fourth International Workshop on Machine Learning*. Elsevier, 1987, pp. 256–266.

[23] A. Khadke and M. Veloso, "Inferring capabilities by experimentation," in *International Conference on Intelligent Autonomous Systems*. Springer, 2018, pp. 889–901.

[24] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[25] P. Andras, L. Esterle, M. Guckert, T. A. Han, P. R. Lewis, K. Milanovic, T. Payne, C. Perret, J. Pitt, S. T. Powers *et al.*, "Trusting intelligent machines: Deepening trust within socio-technical systems," *IEEE Technology and Society Magazine*, vol. 37, no. 4, pp. 76–83, 2018.

[26] G. Lu, J. Lu, S. Yao, Y. J. Yip *et al.*, "A review on computational trust models for multi-agent systems," *The open information science journal*, vol. 2, pp. 18–25, 2009.

[27] S. Edenhofer, S. Tomforde, J. Kantert, L. Klejnowski, Y. Bernard, J. Hähner, and C. Müller-Schloer, "Trust communities: an open, self-organised social infrastructure of autonomous agents," in *Trustworthy Open Self-Organising Systems*. Springer, 2016, pp. 127–152.