Software Engineering for Distributed Autonomous Real-Time Systems

Lenz Belzner[1], Michael Till Beck[1], Thomas Gabor[1], Harald Roelle[2], Horst Sauer[2]
[1] Ludwig-Maximilians-Universität München
{belzner,michael.beck,thomas.gabor}@ifi.lmu.de
[2] Siemens AG München

ABSTRACT

We discuss key challenges of software engineering for distributed autonomous real-time systems and sketch relevant solution approaches.

CCS Concepts

•Computing methodologies → Distributed artificial intelligence; •Computer systems organization → Embedded and cyber-physical systems; •Software and its engineering → Software design engineering; Software verification and validation;

Keywords

software engineering, distributed autonomous real-time systems, verification & validation, non-functional requirements

1. INTRODUCTION

Many industrial applications of the present or near future heavily rely on the use of computers and thus software to be realized. Among those are developments like autonomous factories or self-adjusting power grids (so-called 'smart grids'). Both are characterized by the fact that they build upon standard industry hardware but use elaborate software coordination to generate added value for the user of said systems [1]. As the coordination performed by software may out-perform a human planner, smart systems are expected to function more flexibly and efficiently than the traditional ("not so smart") systems used before. However, as these systems need to interact directly with the physical world and changing business requirements, complexity and continuous change are key properties of domains that modern software systems operate in. Uncertainty and unexpected events and highly volatile business requirements yield new challenges when developing, running and assessing such systems. Due to missing information at design time and the complexity of potential changes occurring at run-time, it is mandatory to provide high-level specifications of the system. Based on these high-level specifications, systems need to be able to autonomously learn about their environment and plan their actions wrt. information only available at runtime, in order to be able to cope with the variety of different situations they may face in complex environments. For example, an autonomous factory could know about all variants of products it is able to produce. At runtime, individual production machines decide collaboratively which specific variants of which products to actually produce, based on current product requests and available production resources.

While some kinds of change can be dealt with by a system that autonomously identifies an adequate reaction at runtime, other changes will require engineers to change the system itself while operating. For example, a system may detect that its specified model is no longer concordant with its latest runtime observations. Also, intended changes may happen to the system by adding or removing components, for example a new transportation machine in an autonomous factory. Software architectures and engineering methods have to support engineers when designing and maintaining systems that are subject to these kinds of change at runtime.

Modern software systems also are built from numerous independent components which are collaborating to solve common tasks or competing for resources. This high degree of distribution adds another layer of complexity to software design for such a system. Allowing for autonomous management of knowledge exchange and interaction between components yields high flexibility, but also gives raise to engineering challenges: Tasks have to be decomposed autonomously at runtime based on information only available locally to individual components. At the same time, the communication and interaction structures have to be optimized according to the particular situation. And finally, potentially colliding goals and interests of participants have to be aligned with global system performance requirements. Scenarios such as planning power supply in smart grids add another aspect of decentralization: The system has only partial control about the entities it has to interact and collaborate with. In such open multi-agent systems – in fact, societies of systems – it is no more feasible to presuppose a collaborative attitude of interacting entities [2]. Like their natural counterparts, societies of systems will have to provide and maintain mechanisms and rules for dealing with malfunctioning or even adversarial participants.

From a system architect's point of view, there is no obvious method available to meet the complexity of modern

Authors' draft. Final version: Belzner, Lenz, et al. "Software engineering for distributed autonomous real-time systems." Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems. ACM, 2016. http://dl.acm.org/citation.cfm?id=2897040

DOI: 10.1145/2897035.2897040

application domains such as smart grids or autonomous factories, which we use as our sample scenarios. It has been recognized in the scientific community that software engineering for these systems poses several difficulties that have not to their full extent been analyzed by the traditional field of software engineering [3] and thus these new systems of interest have been given a variety of new names to differentiate them from more standard applications. Relevant literature can be found under names such as "complex adaptive systems" [4], "ensembles" [3], or "cyber-physical systems" [5], often describing similar phenomena. We propose the notion of *distributed autonomous real-time systems* (DARTS) to explicitly refer to central distinguishing characteristics of such systems. The rest of this paper discusses specific issues regarding software engineering for DARTS: Figure 1 depicts a categorization of these aspects. Section 2 briefly outlines two typical application scenarios of DARTS. In Section 3 we focus on *system autonomy*, i.e. the ability of a system to make meaningful decisions without human intervention. In Section 4, we outline how autonomous *distribution and coordination* of tasks and component interaction enables building scalable and efficient systems. Autonomy of individual components and collective systems yields particular challenges for *verification and validation* of DARTS, which we discuss in Section 5. We conclude in Section 6.

2. SCENARIOS

This section shortly describes two domains of DARTS, serving as examples in the following sections.

Smart Grid In a traditional power grid, power is generated by few, central power plants. These power plants are owned, managed and controlled by the energy provider; the amount of power generated by these plants is adapted to the power demand of the consumers. In a smart grid, however, many distributed power plants become part of the grid. These power plants rely on the availability of renewable power sources, and therefore can not be planned and controlled like traditional power plants. One key challenge the energy provider has to face is to ensure that the amount of power generated by all power plants (even the highly fluctuating ones) matches the amount of power that is consumed – otherwise, the grid collapses. For economic reasons, this should be done in a cost-efficient way. Forecasting and planning play a key role in smart grid scenarios as energy providers have to estimate how much power will be needed and how much power can be provided within the near future. This is usually implemented with the help of weather forecasting services, modeling techniques, and historical data [6]. All of this needs to be performed with a relatively short reaction time while considering a huge amount of producers and consumers connected to the grid, which makes it necessary for the system to make autonomous decisions (i.e., to not ask a human if possible) under time constraints.

Smart Factories Up to now, production processes in factories are static: there is not much flexibility in the production process, as one factory is usually built for producing one specific type of product with only few possible variations. Robots are controlled and managed by a centralized entity and are rarely reconfigured. Contrary, in smart factories robots are interconnected and equipped with powerful capabilities (e.g., computing power, networking capabilities, sensors, and physical interfaces). Production is a distributed process: robots operate autonomously and are able to manage and adapt production processes collaboratively. Production becomes much more adaptive to market demands and enables new products which can be highly personalized to the specific needs of the customer with the products still being economically feasible and quickly available.

3. AUTONOMY ENGINEERING

Systems such as smart grids or autonomous factory robots have to deal with high degrees of uncertainty and change. In a smart grid, the weather is a probabilistic factor with large influence on the task to solve; other grid participants may give wrong assertions about their power supply and consumption, be it intentional or not; and energy supply and consumption can typically only be estimated more or less accurately. Autonomous factories have to deal with constantly changing production requests, human collaborators that interact with them in a shared environment, and sensory data that is noisy and incomplete. In both domains, participants may join or leave collaborations. While this happens according to a set schedule in a typical factory, participants may do so completely unexpectedly in a smart grid.

3.1 Approach

While it is one option to manually build and improve models and behavior specifications for highly dynamic and complex domains of this kind, it seems a more useful alternative to provide more abstract specifications to a system. This abstraction in turn allows a system to use information available at runtime (e.g. sensory data) in order to instantiate concrete environmental models and to plan and execute actions accordingly.

System synthesis is typically done by optimization wrt. to a given goal function. This property has an appealing side effect: It allows system operators to change behavior at runtime by configuring the objective function, not changing any other part of the system. While the running system may not be optimally equipped to fulfill a new task (e.g. a factory robot without a magnetic gripper is required to operate with metallic materials), system autonomy will still yield adequate behavior (e.g., the robot reverts to its available non-magnetic gripper).

Since DARTS operate in real-time, it is necessary to generate decisions within given time limits. Additionally, as DARTS are decentralized autonomous systems, good timing may be even more crucial than that of pure performance. For example, it may be necessary for two machines in an autonomous factory to work on a single piece of material at a coordinated point in time. Furthermore, there is a trade-off between planning time reduction and plan quality gain. Specifying the optimal range for this trade-off is a crucial challenge for DARTS. Effectively, a certain level of performance is required from the learning techniques to even function; we expect the notion of performance in DARTS to comprise both time and quality. From another perspective, the performance of an adaptive system can be expressed by the amount of information it can process per time span.

We argue that system autonomy is a necessary property of DARTS, as their complex, dynamic and uncertain environments urge systems to self-adjust.

3.2 Foundations

In line with related work [2], we consider three foundational areas of system autonomy: *Abstraction*, *learning* and

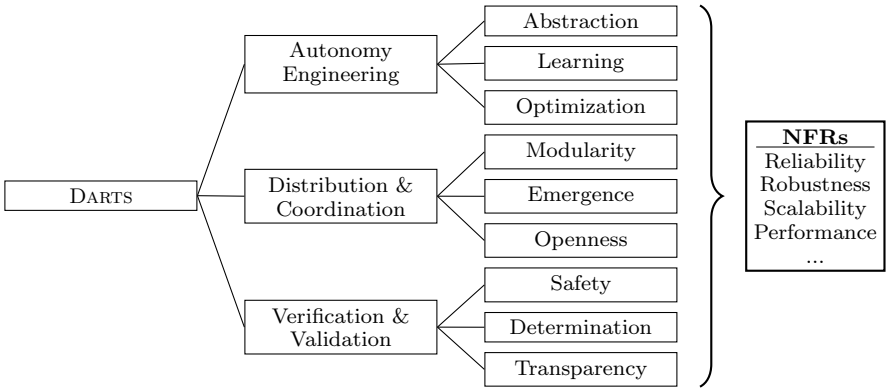

Figure 1: Autonomy engineering, distribution & coordination as well as suitable verification & validation are preliminary to ensure quality of Darts operating in complex domains.

optimization (cf. Fig. 1). As these are mutually dependent, we also identify their integration as one of the key aspects of system autonomy.

Abstraction System synthesis is highly sensitive to the particular abstraction of a problem (i.e. its representation). Representation learning allows to extract relevant abstractions from low-level data (e.g. sensory data). Recently, representation learning extracted useful abstractions for tasks such as digit recognition or speech synthesis [7,8]. Darts operate in partially unknown and changing domains. Enabling them to build abstractions at runtime in a task-specific manner yields the potential to greatly impact model and behavior synthesis without requiring manual feature engineering. In smart factories, these techniques can be applied to learn a model of the correlation between throughput and quality of the items produced in a smart factory, which can then be used by the smart factory to adjust to changing quality or delivery requirements. In a smart grid, abstraction plays a role in deriving the current global state of the grid based on a large, incomplete, and possibly imprecise set of data (related to the term *forecasting*, this process is also known as *nowcasting* [6]). In both cases, the abstraction process decides which data are considered important for the observation module and how data need to be transformed and combined in order to derive relevant information.

Learning Darts should be able to compile runtime information to models (i.e. knowledge) of the environment. Techniques include decision forests [9], Gaussian processes [10], or other machine learning techniques [11,12]. Learning models from runtime data allows (a) to make informed decisions even in the absence of a-priori specified models, (b) to detect and react to changes of domain dynamics by statistically assessing runtime observations over time and (c) to detect and recover from inconsistencies of specification and runtime observations by comparing accuracy of learned and already existing models.

For example, an autonomous factory robot could be able to move product parts and screw two of them together and communicate said functionality to other agents in the factory. When the screwing mechanism of the robot fails, it can learn about that change in its physique by observing the results it produces or listening to messages delivered from robots that try to further work on the misfit parts. It can then update its behavior to refrain from trying to join parts and fall back to moving the parts only. In the smart grid, model learning could enable adaptation to changing weather conditions or prediction of other participants' behavior.

Furthermore, models describing the behavior of plants like photo-voltaic power plants can be learned, refined and replaced at runtime, as the efficiency of those plants diverges, depending on parameters like temperature, age, inclination angle, and others [6]. The same applies to models describing power consumption behavior of specific consumers or groups of consumers: The system is able to detect and learn patterns of power consumption behavior (e.g., a large smart factory consumes most power at noon, but only on weekdays) and adapts them accordingly if changes occur. In contrast to the observation module, models derived by the learning process are not used for nowcasting the current state of the system, but for forecasting effects of possible future actions to the system state.

Optimization Robustness and reliability are achieved by using learned abstractions and models for planning and decision making, optimizing system behavior and configuration wrt. current situation and system goals. In general, online optimization is a useful technique to improve a system's ability to fulfill certain non-functional requirements once a prototype fulfilling all functional ones is built. E.g., a smart factory can be given a specific list of operations (in a very primitive form, this can be thought of as something like "join, drill, paint") that ensure a product is built according to specification. If the smart factory notices that it has many free drills available but few transport robots able to lift the heavier joint parts, it may be advantageous to change that plan to "drill, join, paint", thus alleviating the transport robot bottleneck. Thus, the system can (and to some extent, is expected to) increase its own performance over time. In a smart grid, weather forecasts could be taken into account for planning operational schedules of power plants for the next day(s). Also, predictive models learned from historical data describing power consumption behavior of specific parts of the grid and larger consumers could be used as a source of information for planning.

Model-based reinforcement learning with open-loop planning can drive the reasoning and decision process in the changing, dynamic domains of Darts [13–15]. In general, optimization for very large, continuous problems can be tackled by meta-heuristic search. Sampling theory provides a principled approach for efficient use of resources available for optimization, e.g., by using particle filter approaches and

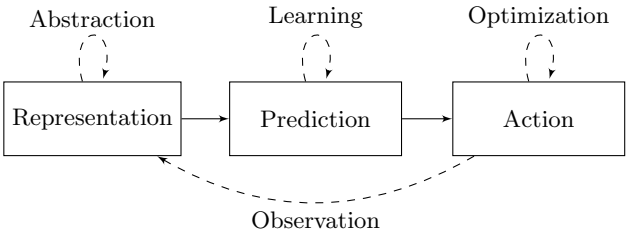

Figure 2: Interplay of Abstraction, Learning and Optimization (adapted from [2])

importance sampling techniques [16, 17], or evolutionary algorithms [18].

Integration Figure 2 illustrates the mutual interplay of abstraction, learning and optimization: These components of system autonomy expose clear dependencies, as the output of each serves as input for another – representational abstraction drives learning, learning drives acting, and acting yields observations that influence representational abstraction and/or the learning process. E.g., in a smart grid actions taken based on error-prone models could lead to unfavorable states. For instance, the weather forecasting service may communicate divergent data (as those forecasts are usually estimated and derived from measurements taken from sensors deployed within the larger geographical region), or the power consumption behavior of consumers may have changed. A central challenge when engineering DARTS will be to provide transparent and comprehensible ways to identify and illustrate this interplay, be it to support system design or to better understand the autonomous adaptation process. See e.g. [19, 20] for recent results of integration of representation learning and decision making.

4. DISTRIBUTION AND COORDINATION

Modern tasks such as energy planning or autonomous manufacturing pose complex challenges to the software systems designed to tackle them. Distributed system architectures enable the creation of complex behavior by composition of simple components, enabling scalable and flexible system behavior. Using distributed architectures also provides robustness against failure of single agents. Furthermore, parallelization of tasks can – if coordinated appropriately – increase system performance dramatically. However, communication overhead can also have detrimental effects on system performance. Thus, oftentimes there exists a certain trade-off between information distribution and information exploitation w.r.t. common performance measurements.

In an autonomous factory, assignment of machines to particular tasks (e.g., transportation of materials, drilling, polishing, etc.) could be dynamically adjusted accounting for current production needs. To render this assignment feasible also for a large number of machines that potentially have only local information (e.g., materials that are passed to them, or information that nearby machines shared), task assignment and collaborative structure have to be determined in distributed manner. In a smart grid, the same idea of scalability applies: For thousands of dynamic participants it is often not possible (and certainly not feasible) to compute supply and consumption reliably and efficiently. Instead, the central problem of global energy balance can be tackled by allowing the system to autonomously divide and conquer the task in accordance to current circumstances.

4.1 Approach

Autonomous distributed systems are theoretically rooted in game theory and its variants, e.g. evolutionary game theory. While classic game theory is based on the idea of rational agents (i.e. individuals that only want to maximize their own welfare), modern variants allow to model and deal with more subtle aspects of social interaction.

Multi-agent systems can be seen as software incarnation of game-theoretic considerations. They are widely used, and characteristically consist of distributed, yet interacting individual decision makers. Multi-agent approaches are recently gaining much attention in the domain of smart grid management and optimization [21–23].

Distribution is not only relevant on the level of interaction structure, but also when decomposing global tasks in local ones that are more feasible to be solved. Distributed constraint optimization allows for scalable and efficient multi-agent decision making [24]. Parallelization of planning algorithms such as Monte Carlo Tree Search also provides gains in terms of performance and robustness [25, 26].

Finding behavioral compromises to balance local and global goals of systems and individuals alike is a central issue of coordination in distributed systems. Enabling autonomous construction of such compromises is central to successful engineering of DARTS. In the area of multi-agent systems, there exists ample research on how to reach such a group consensus [27–29]. The framework of probability collectives is designed for probabilistic collective decision making based on importance sampling. Bio-inspired approaches to model and control emergent behavior have a strong impact on multi-agent systems. For example, swarm-structures exhibit many properties valuable from an engineering point of view, such as scalability, robustness, modularity and simplicity of individual components.

Trust-based computing enables agents and systems to build and maintain confidence about entities they have to interact with. For example, a malfunctioning agent may repeatedly make promises about its capabilities, on which the collective bases its decisions. The malfunctioning agent does not keep its promise, causing a decline in system performance. Trust values allow the system to identify this defect and to generate an adequate reaction, e.g. not weighting the particular agent's promises to strongly when finding collective decisions. Trust in certain planning agents can also be used as a coordination mechanism for the distribution of plans in DARTS [30].

Social norms are a valuable engineering tool in order to reduce the space of possibilities when interacting with other agents, and in order to allow for feasible predictability of behavior. As in human societies (e.g., typical drivers stop at a red traffic light), these norms provide rule sets on which agents can base their own decisions without intractable overhead. Rules also enable the identification of individuals who do not obey them; distributed systems may then react to them accordingly.

4.2 Foundations

Modularization Modularization plays a key role for DARTS. It takes place on different levels: (a) The design of DARTS should provide modular components that can play different

roles dynamically, adaptively adjusting their current behavior driven by software. Components can be dynamically added or removed from DARTS at runtime. (b) Runtime tasks have to be decomposed into adequate subtasks, which are subsequently assigned to different collaborative components of the system. (c) These collaborative components have to be optimized wrt. the task at hand. Dynamic modularization of components, tasks and structures are mutually coupled problems: For example, an adequate task decomposition depends on a system's communication and collaboration structure. However, this structure should be optimized in accordance with the identified subtasks as well.

A key to enabling scalability via modularization may be the decomposition of tasks or structure wrt. their particular spatial and temporal characteristics. Spatial task allocation problems have recently been applied to distributed online planning based on spatial decomposition in robotics [31]. As the approach has yet only been applied to a discrete domain, a direct extension of this work would be to leverage the approach to continuous domains, thus enabling real-time modeling and planning for DARTS. Another direction could investigate task distribution and solving while accounting for temporal task properties.

Emergence Another central issue of distributed autonomy is emergence. Local agent optimization wrt. their individual knowledge and goals yields global interaction patterns and effects. An understanding of the interconnection of local changes and global impact is preliminary for controlling emergent system behavior. Also, when agents act in their own interest, it is important to specify to what extent their runtime decision should account for global welfare. For instance, regional power grids are usually interconnected to other grid networks for safety reasons: in case of a local power shortage or power surplus, power can be bought from or sold to those external grid providers. In this context, the term emergence refers to the global network of networks (the global grid as a network of regional grids). Each regional grid is controlled and managed by the local energy provider. Based on contractual agreements, each regional provider ensures the adherence of certain guarantees, resulting in global emergence of the system. However, emergent phenomena may also be detrimental to the system's performance, e.g., when a large factory with too many robots spends too much time exchanging messages and coordinating routes and does not manage to actually produce enough items. In this case, scalability of the system is lost due to emergence.

Openness A third challenge arises from the fact that DARTS are designed to interact with open environments. External, uncontrolled agents and participants pursuing their own goals (e.g., human collaborators in a partially autonomous factory) require DARTS to adaptively adjust their behavior. These external entities may be accidentally or intentionally counterproductive or even malicious. Also, they may enter or leave a domain at any time and at their own will. For example, in a smart grid, the difficulty in maintaining a suitable power supply may change drastically depending on which providers and consumers participate in the grid at the moment. In a smart factory, external producers may deliver parts that are half-processed and only need to go through specific parts of the factory's production process while skipping others. An especially smart design or adaptation of the factory may even prepare for this case before the half-ready items even enter the production line. Taking this openness of problem domains into account is a crucial requirement to successfully develop and run DARTS.

5. VALIDATION AND VERIFICATION

Abstract system specification—e.g. by probabilistic domain models and algorithms for system synthesis—enables systems to configure and decide based on deliberation at runtime, and intentionally defines probabilistic factors and uncertainty. Requirement specification should explicitly account for this uncertainty. For this purpose, it is often helpful to define continuous measurements of solution quality that can be used for automated optimization. The transformation of goal specifications to utility functions that are optimized by DARTS at runtime plays a central role of requirements engineering and system design. Also, it should be stated clearly which parts of the requirements are hard constraints (e.g. safety requirements), and which ones are to be optimized by the system at runtime.

Typically, not a single solution will be specified, but a whole solution space: Which concrete solution is chosen depends on runtime information and autonomous system optimization. The same principle should apply to test cases for DARTS: Tests will be specified abstractly, allowing for a concise definition of possible runtime situations. Concrete instantiations of tests can potentially be generated based on well-defined parameters (e.g., the number of robots in a factory, or different weather conditions in the smart grid). Assessment of DARTS will then be of statistical nature, as not all situations can be explicitly captured by a test suite due the vast number of possible scenarios. However, techniques such as co-evolution of test cases can be used to identify critical regions of the test space by refining test cases that are hard for the autonomous system to deal with.

The level of abstraction used for the specification of DARTS enables adaptive system autonomy, but also yields a number of challenges for verifying and testing: (a) Runtime system decisions should be transparent and explicated to enable iterative control of system design. (b) Effort, cost and risk of assessment should be minimized. (c) Hard constraints must not be violated. In the following, we will discuss approaches to these challenges.

5.1 Approach

While abstract specification of system configuration, behavior and test spaces allows to deal with highly complex domains, some situations require absolute safety – at least from the side of the system. For example, DARTS for autonomous factories must not injure human collaborators. While it is not possible to rule out human mistakes, it is possible to provide behavioral guidelines for humans for which guarantees can be given. Another approach is to use deterministic, rule-based safety mechanisms (as in current systems) and add components that autonomously identify critical situations that have not been recognized as such by the original safety system. For example, a factory robot may be equipped with a component that detects humans and halts the machine in case someone is getting too close. However, autonomous learning and planning may provide additional safety by providing behavioral strategies for cases where the detection component fails for whatever reason. Designing and evaluating such 'watchdog' architectures is a central challenge for successful engineering of DARTS.

Virtualization plays a central role when minimizing effort,

cost and risk of DARTS assessment. Modern simulation software provides highly efficient and tunable toolsets for setting up and running DARTS in various scenarios. DARTS are designed to operate in open environments with a large proportion of dynamics that are not under control of the system. Also, state and action spaces grow extremely large, if not to infinity. This renders exhaustive model-checking approaches infeasible. To this end, quantitative analysis and statistical approaches to system validation and verification are central to ensure quality of DARTS. Modern simulation software provides the capability to (a) define complex test scenarios with a minimum of effort (e.g. cost, risk), incorporating (b) highly realistic sources of data, such as visual, auditive or other sensory information or physical simulation data that is very close to real data [33, 34]. Another benefit of simulation is that it can be used for analyzing DARTS at design time, but also provides virtual system representations that can be used to drive system autonomy and optimization at runtime (c.f. Section 3). In order to allow for a structured assessment, requirement specifications have to capture the autonomous, distributed character of DARTS. Interestingly, many online decision and optimization algorithms are based on sampling a virtual model of the system. In fact, the same arguments – i.e. minimization of effort, cost and risk – motivate the use of simulation and virtualization for online planning. It seems valuable to exploit and further investigate this duality for engineering of DARTS.

Autonomous system configuration and behavior synthesis based on distributed information yields emergent phenomena that system designers may wish to enhance or eliminate. For example, autonomous distributed planning in autonomous factories may expose particularly helpful or rather useless collaborations of agents that have not been anticipated by engineers. When engineering DARTS, these configurations should be (a) identified, (b) evaluated, (c) integrated into the development process and (d) their causes should be analyzed and documented. In particular, transparency of autonomous system decisions plays a key role for these tasks. Therefore, it is crucial to integrate techniques for data visualization and explication of system decisions as early as possible in the development process.

5.2 Foundations

Safety One way to ensure system correctness and to guarantee safety properties are model checking approaches. Model checking can also be used to generate system traces for analysis, fostering transparency of system decisions. Requirements are encoded as formulas in some logic; if a system satisfies these formulas, it is said to meet the requirements. [35] provides a comprehensive overview to logic-based approaches for specification and verification of multi-agent systems. Various software tools exists to support formal validation and verification: PRISM is a model checker offering implementations of a variety of different techniques; PRISM supports systems with both probabilistic and real-time behavior [36]. PRISM-GAMES provides support for stochastic multi-player games, allowing for abstract formal modeling and verification of collaborative and competitive distributed system behavior [37]. UPPAAL is an environment for modeling, validating, and verifying real-time systems [38]. The Markov Reward Model Checker MRMC enables reachability checks under time constraints; it supports discrete-time and continuous-time Markov chains [39].

As mentioned above, tight integration of deterministic components with system autonomy on an architectural level seems a promising direction to enable safety guarantees for DARTS. However, to the best of our knowledge no generically applicable approach for (multi-)agent safety that incorporates restricted design time knowledge and system openness at runtime has been described in the literature so far.

Determination From a system theory perspective, DARTS navigate through a phase space, with each point in the phase space representing a theoretically possible state of the system. In general, predicates regarding properties of the system's state can be associated with regions (i.e. sets of points) in the phase space. For example, hard safety requirements can be mapped to a corridor in the phase space we forbid the system from leaving [40]. Contrarily, continuous goal definitions can be viewed as an adaptive fitness landscape spanning over the phase space, i.e., if it is possible to do so without violating safety requirements, we expect the system to wander towards better system states according to its goal definitions. This behavior ensures that the system tries to fulfill its goals as well as possible, resulting in a certain facet of liveliness. As an example, a smart factory doing exactly nothing at all is quite certain to fulfill all its safety requirements. However, it is not *determined* to fulfill its foremost goal (producing goods effectively) and thus pretty useless.

Since we expect DARTS to optimize their behavior autonomously, it is important to note that determination can be often better understood as a property of the underlying learning mechanisms, since it is them which control the dynamic system behavior. For learning algorithms, determination can be observed as variance reduction: Without major changes to environmental circumstances, the learning algorithm should focus its efforts on increasingly promising regions of the search space. Monte Carlo search techniques based on importance sampling like MCTS or the cross entropy method implement this behavior by sampling more promising scenarios with higher probability [13–15].

Transparency Search-based software testing (SBST) aims at generating informative test cases for systems with large input spaces based on meta-heuristic search [41, 42], sampling theory and strategies such as importance sampling [17] or co-evolution of system configurations and test-cases [43]. SBST provides a principled approach to deal with the state space explosion of DARTS in the context of validation, verification and testing. For a recent discussion of optimal experimental design aiming at generating maximally informative data in simulations see [44].

6. CONCLUSION

We discussed key aspects of software engineering for distributed autonomous real-time systems. We argued that enabling system autonomy ensures that DARTS operating in highly dynamic, open environments that are only partially known at design time can meet non-functional requirements such as robustness, flexibility, scalability and efficiency.

Individual agent autonomy allows to synthesize models and behavior based on runtime information, reducing the impact of design time uncertainty. Also, autonomous task distribution and system coordination yields flexible yet scalable systems able to cope with large, highly dynamic tasks.

Enabling system autonomy to ensure quality of DARTS in complex domains yields new challenges for their validation

and verification. We argued that virtualization plays a central role both in terms of scalability to domain complexity and system behavior transparency and explication. Formal approaches such as model checking, as well as combination of deterministic and autonomous techniques and specialized architectures provide an interesting perspective to enable safety guarantees for DARTS.

Acknowledgements

This research has been supported by SIEMENS AG Munich, Germany.

7. REFERENCES

[1] J.-P. Banatre and M. Hölzl, *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*. Springer Science & Business Media, 2008, vol. 5380.

[2] L. Belzner, M. Hölzl, N. Koch, and M. Wirsing, "Collective autonomic systems: Towards engineering principles and their foundations," 2015, submitted for publication.

[3] M. Wirsing, M. Hölzl, N. Koch, and P. Mayer, *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Springer, 2015, vol. 8998.

[4] J. H. Holland, "Studying complex adaptive systems," *Journal of Systems Science and Complexity*, vol. 19, no. 1, pp. 1–8, 2006.

[5] T. Bureš, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "The invariant refinement method," in *Software Engineering for Collective Autonomic Systems*. Springer, 2015, pp. 405–428.

[6] M. T. Beck, S. Schuster, H. De Meer, and M. Kreuzer, "Estimating photo-voltaic power supply without smart metering infrastructure," in *Energy-Efficient Data Centers*, ser. Lecture Notes in Computer Science, S. Klingert, X. Hesselbach-Serra, M. Pérez, and G. Giuliani, Eds. Springer Berlin Heidelberg, 2014, vol. 8343, pp. 25–39.

[7] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[8] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, and G. E. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder." in *Interspeech*. Citeseer, 2010, pp. 1692–1695.

[9] A. Criminisi, J. Shotton, and E. Konukoglu, *Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning*. Now, 2012.

[10] C. E. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[11] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[12] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[13] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[14] A. Weinstein and M. L. Littman, "Open-loop planning in large-scale stochastic domains," in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, M. desJardins and M. L. Littman, Eds. AAAI Press, 2013.

[15] L. Belzner, R. Hennicker, and M. Wirsing, "Onplan: A framework for simulation-based online planning," in *Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers*, 2015, pp. 1–30.

[16] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan, "The unscented particle filter," in *NIPS*, 2000, pp. 584–590.

[17] R. Srinivasan, *Importance sampling: Applications in communications and detection*. Springer Science & Business Media, 2013.

[18] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, second edition ed. Springer, 2007.

[19] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015, pp. 1766–1774.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[21] P. Agrawal, A. Kumar, and P. Varakantham, "Near-optimal decentralized power supply restoration in smart grids," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 2015, pp. 1275–1283.

[22] J. Cerquides, G. Picard, and J. A. Rodríguez-Aguilar, "Designing a marketplace for the trading and distribution of energy in the smart grid," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, Eds. ACM, 2015, pp. 1285–1293.

[23] A. Angelidakis and G. Chalkiadakis, "Factored MDPS for optimal prosumer decision-making," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 2015, pp. 503–511.

[24] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *Artificial Intelligence*, vol. 161, no. 1, pp. 149–180, 2005.

[25] G. M.-B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *Computers and Games*. Springer, 2008, pp. 60–71.

[26] R. B. Segal, "On the scalability of parallel uct," in *Computers and Games*. Springer, 2011, pp. 36–47.

[27] S. Li, H. Du, and X. Lin, "Finite-time consensus algorithm for multi-agent systems with

double-integrator dynamics," *Automatica*, vol. 47, no. 8, pp. 1706–1712, 2011.

[28] G. Wen, Z. Duan, W. Yu, and G. Chen, "Consensus in multi-agent systems with communication constraints," *International Journal of Robust and Nonlinear Control*, vol. 22, no. 2, pp. 170–182, 2012.

[29] G. Wen, Z. Duan, G. Chen, and W. Yu, "Consensus tracking of multi-agent systems with lipschitz-type node dynamics and switching topologies," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 2, pp. 499–511, Feb 2014.

[30] M. Hölzl and T. Gabor, "Continuous collaboration: A case study on the development of an adaptive cyber-physical system," in *Proc. of the International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), Firenze, Italy*, 2015.

[31] D. Claes, P. Robbel, F. A. Oliehoek, K. Tuyls, D. Hennes, and W. van der Hoek, "Effective approximations for multi-robot coordination in spatially distributed tasks," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 2015, pp. 881–890.

[32] S. D. Ramchurn, T. D. Huynh, Y. Ikuno, J. Flann, F. Wu, L. Moreau, N. R. Jennings, J. E. Fischer, W. Jiang, T. Rodden, E. Simpson, S. Reece, and S. J. Roberts, "HAC-ER: A disaster response system based on human-agent collectives," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 2015, pp. 533–541.

[33] M. Pharr and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.

[34] I. Millington, *Game physics engine development*. Morgan Kaufmann Publishers Amsterdam, 2007.

[35] M. Dastani, K. V. Hindriks, and J.-J. Meyer, *Specification and verification of multi-agent systems*. Springer Science & Business Media, 2010.

[36] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[37] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "PRISM-games: A model checker for stochastic multi-player games," in *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, ser. LNCS, N. Piterman and S. Smolka, Eds., vol. 7795. Springer, 2013, pp. 185–191.

[38] N.N., "UPPAAL," http://uppaal.org/, accessed: 2015-07-02.

[39] ——, "MRMC," http://www.mrmc-tool.org/trac/, accessed: 2015-07-02.

[40] D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli, "Sota: Towards a general model for self-adaptive systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. IEEE, 2012, pp. 48–53.

[41] P. McMinn, "Search-based software test data generation: a survey," *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[42] ——, "Search-based software testing: Past, present and future," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, March 2011, pp. 153–163.

[43] A. Assis Lobo de Oliveira, C. Gonyalves Camilo-Junior, and A. Vincenzi, "A coevolutionary algorithm to automatic test case selection and mutant in mutation testing," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, June 2013, pp. 829–836.

[44] X. Huan and Y. M. Marzouk, "Simulation-based optimal bayesian experimental design for nonlinear systems," *Journal of Computational Physics*, vol. 232, no. 1, pp. 288–317, 2013.